NINEVAH UNIVERSITY

COLLEGE OF ELECTRONICS ENGINEERING

COMPUTER AND INFORMATION

ENG. DEPARTMENT

Visual Impaired Assistance System Based on YOLOv3 Network

By

Raghad Raied Mahmood

M.Sc. Thesis

In

Computer and Information Engineering

Supervised by

Dr. Majid Dherar Younus
Dr. Emad A. Al-Sabawi

2021 A.D. **1443 A.H.**

Visual Impaired Assistance System Based on YOLOv3 Network

A Thesis Submitted

By

Raghad Raied Mahmood

To

The Council of the College of Electronic Engineering

Ninevah University

As a Partial Fulfillment of the Requirements

For the Degree of Master of Science

In

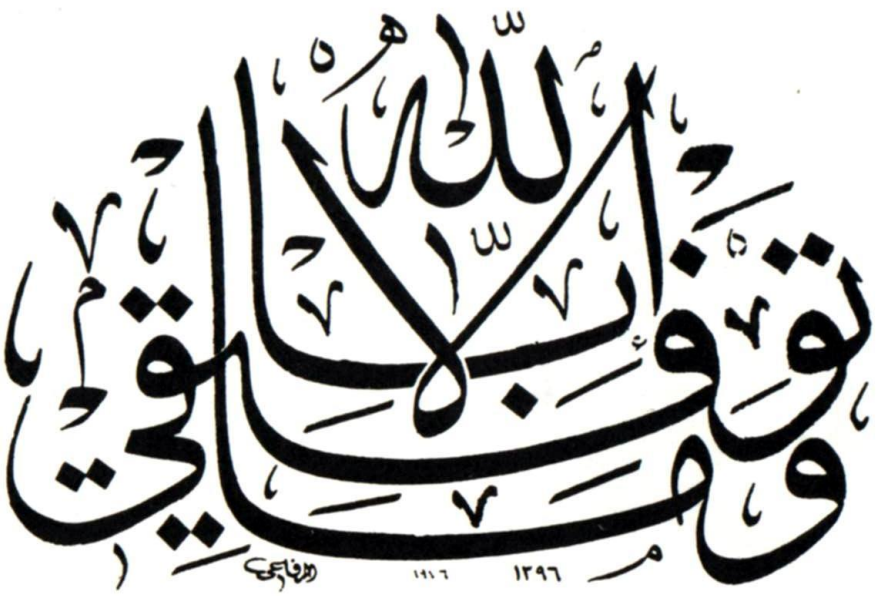
Computer and Information Engineering

Supervised by

Dr. Majid Dherar Younus
Dr. Emad A. Al-Sabawi

2021 A.D. **1443 A.H.**

بسم الله الرحمن الرحيم



صدق الله العظيم

سورة هود

الآية (88)

Supervisor's Certification

We certify that the dissertation entitled (**Visual Impaired Assistance System Based on YOLOv3 Network**) was prepared by **Raghad Raied Mahmood** under our supervision at the Department of Computer and Information Engineering, University of Ninevah, as a partial requirement for the Master of Science Degree in Computer and Information Engineering.

Signature: Signature:

Name: Dr. Majid Dhrar Younis **Name: Dr. Emad A. Al-Sabawi**

Date: / /2021

Linguistic Advisor Certification

I certify that the linguistic evaluation of this thesis entitled **"Visual Impaired Assistance System Based on YOLOv3 Network"** was carried out by me and it is accepted linguistically and in expression.

Signature:

Name: Asst. Lecturer Mahmood Ahmed Mahmood

Date: / /2021

Post-Graduate Committee Certification

According to the recommendations presented by the supervisor of this dissertation and the linguistic reviewer, I nominate this dissertation to be forwarded to discussion.

Signature:

Name: Assistant Prof. Maan A. S. Al-Adwany

Department Head Certification

I certify that this dissertation was carried out in the Department of Computer and Information Engineering. I nominate it to be forwarded to discussion.

Signature:

Name: Assistant Prof. Maan A. S. Al-Adwany

Date: / /2021

Committee Certification

We the examining committee, certify that we have read this dissertation entitled (**Visual Impaired Assistance System Based on YOLOv3 Network)** and have examined the postgraduate student (**Raghad Raied Mahmood)** in its contents and that in our opinion; it meets the standards of a dissertation for the degree of Master of Science in Computer and Information Engineering.

Signature:
Name: Prof. Dr. Shefa Abdulrahman Dawwd
Head of committee
Date: / /2021

Signature:
Name: Asst. Prof. Dr. Mohammed H. Al-Jammas
Member
Date: / /2021

Signature:
Name: Asst. Prof Dr. Mohammed Abdul-Muttaleb Abdullah
Member
Date: / /2021

Signature:
Name: Dr. Majid Dhrar Younis
Member and Supervisor
Date: / /2021

Signature:
Name: Dr. Emad A. Al-Sabawi
Member and Supervisor
Date: / /2021

The college council, in it's …… meeting on / /2020, has decided to award the degree of Master of Science in Computer and Information Engineering to the candidate.

Signature:
Name:
Dean of the College
Date: / /2021

Signature:
Name:
Council Register
Date: / /2021

ACKNOWLEDGEMENTS

Praise to ALLAH Al-Mighty, Lord of the World. Praise be to ALLAH who has given me strength and ability to complete my thesis.

I would like to express my sincere gratitude and thanks to my supervisors, Dr. **Majid Dherar Younus** and **Dr. Emad A. Al-Sabawi** for their continuous guidance, useful suggestions, constant encouragement and assistance to me throughout the research period.

I also express my gratitude and thanks to the head and all members of the Computer and Information Engineering Department who have provided me with science and knowledge throughout the preparatory year.

And thanks to all those who have given me advice and provided me with useful information during the period of the research. I thank my dear parents for their continuous support, my brother and my sisters for standing with me and supporting me throughout my study. Thank also extended to all my friends who have supported me. Last, but not the least, I thank ALLAH Al-Mighty for everything.

Researcher

Raghad Raied Mahmood

2021

ABSTRACT

According to World Health Organization records, there are more than 285 million people who are visually impaired. It is relatively simple for a normal human to deal with objects in the surrounding environment, but it is one of the major problems for visually impaired people. Another issue, money plays an important role in our everyday lives and is required for every business transaction. The current thesis proposes a visual impaired assistance system to help visually impaired people by converting the visual world to audio commands. The system has been developed using deep learning based on YOLOv3 two-stage models. Forty four objects have been chosen to detect using a completely patient-centered approach. A label of the detected object is converted into an audio command using Text to Speech library. The trained model performance has been evaluated on a test dataset and real-time live video. Results show that the proposed system can detect and recognize objects with high mean average precision reaches to 88.585%, and 97.647 % for Iraqi banknotes.

Two embedded platforms have been used in the hardware implementation of the visual impaired assistance system. The first embedded system is the Raspberry Pi 3B and the second is the NVIDIA Jetson Nano. The results of the two platform implementations show that the inference speed of the Jetson Nano is significantly faster than the Raspberry Pi 3B.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Name
ADAM	Adaptive Moment Estimation Optimizer
AI	Artificial Intelligence
API	Application Programming Interface
AUC	Area Under The Curve
AP	Average Precision
BN	Batch Normalization
CL	Convolutional Layers
CNN	Convolutional Neural Network
COCO	Common Objects in Context Dataset
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DENSENET	Dense Convolutional Network
DPM	Deformable Part-Based Model
DNR	Did Not Run
FC	Fully Connected
FN	False Negatives
FP	False Positives
FPGA	Field-Programmable Gate Array
FPN	Feature Pyramid Network
FPS	Frames Per Second

GIOU	Generalized IoU
GPS	Global Positioning System
GPU	Graphics Processing Unit
TTS	Text To Speech
HOG	Histograms of Oriented Gradients
IDE	Integrated Development Environment
IoU	Intersection over Union
LR	Learning Rate
mAP	mean Average Precision
NMS	Non-Maximum Suppression
OIDV4	Open Images Detection Dataset V4
ORB	Oriented Fast And Rotated Brief
P	Precision
R	Recall
RCNN	Region Based Convolutional Neural Networks
RELU	Rectified Linear Unit
RESNET	Residual Neural Network
SDK	Software Development Kit
RPI	Raspberry Pi
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SSD	Single Shot Detector

STT	Speech To Text
SVM	Supported Vector Machine
TF	TensorFlow
TN	True Negative
TP	True Positives
YOLO	You Only Look Once

CHAPTER ONE

Introduction

1.1 Overview

Vision is one of the most important human senses. It is an essential part of human life. About 83% of human knowledge is observed using the eyes [1]. Therefore, it is very difficult for the visually impaired to perform daily tasks and interact with the surrounding world. They have to struggle a lot just to do their day-to-day chores.

Visually impaired people find it challenging to walk, recognize objects, drive, recognize banknotes, or find places alone. For that reason, many visually impaired people will bring a sighted friend or family member to help deal with unknown environments, which makes their lives more difficult than normal people. These social challenges limit a blind person from reaching normal people's ability, and this only adds to their low self-esteem [2] [3].

According to a 2020 World Health Organization (WHO) survey, approximately 285 million people are visually impaired, with 39 million being blind and 246 million having low vision. The number of visually impaired people is exploding with the growth of the newborn population, eye diseases, accidents, aging, and so on. Every year this number grows by up to 2 million worldwide [4] [5]. People often regard the visually impaired as a burden and abandon them to fend for themselves. As a result, the visually impaired individual is continuously in need of an assistive technology that may assist in visualizing the world to live a normal life [6].

The walking cane (also known as a white cane or stick) and guide dogs are the most common and oldest mobility aids for the visually impaired. The white cane was chosen for various reasons, including its low cost, portability, and widespread acceptance in the blind community. However, these assistive have their own limitations when a variety of hurdles and situations arise in their daily life [7].

Through the advancement of technology, humanity now has access to a lot of possibilities that would have been either impossible or miraculous in the past, such as object detection [8].

Object detection can be identified as a technology that imitates the functions of the human visual system to detect objects. Object detection is a technique that detects objects of a particular class in live videos. There are several methods for object detection using computer vision, and some methods are more reliable and robust than others. The most modern method for object detection is deep learning [9].

With the recent advances in deep learning, it is possible to extend the support given to people with visual impaired. As a result, the current thesis will shed light on a system that will assist visually impaired people by detecting objects and providing necessary information about those objects.

1.2 Literature Review

To support the visually impaired people, many technologies have been developed. Some relevant works connected to this segment are described below:

- In 2012, B. Schauerte et al. [10], presented a computer vision system that helps blind people find lost objects. It uses color and scale invariant feature transform (SIFT) based on object detection with signification to guide the hand of the user towards potential target object location. This method allows the user's attention to be directed while effectively reducing the amount of space in the environment that needs to be explored.
- In 2014, M. Nalawade et al. [11], implemented a system on an android device to help blind people navigate without the help of a third person. A webcam captures images in front of blind people, then the captured image will be processed through algorithms, which will enhance the image data. The hardware component will have its own database, and the processed image is compared with the database in the hardware component. The result after processing and comparing will be converted into speech signals which used to guide blind people.
- In 2016, P. Jadhav et al. [12], implemented a system to help the visually impaired people using a simple android application. The goal of this system was to search for objects in the camera view using object detection and feature extraction. The authors provided an idea to implement the oriented fast and rotated brief (ORB) algorithm on a field-programmable gate array (FPGA) to increase the execution speed by utilizing the reconfigurable nature and pipelining of the FPGA.
- In 2017, J. Zraqou et al. [13], presented a method for assisting blind people using object recognition. To give the essential information about the surrounding environment, two cameras are installed on a blind person's glasses, a free GPS service, and an ultrasonic sensor are used. The GPS service is used to group objects based on their locations, and the sensor detects any obstacle at a medium distance.

- In 2017, Pachhaiammal [14], proposed a system using a camera to capture the image, then the image will be processed to extract the features of the object using two algorithms. The simulation results using the SIFT algorithm and the Sobel algorithm are used to extract edge key points matching that showed good accuracy for detecting objects. The captured image will be compared with the images that are already stored in the datasets. The images that are matched with the dataset images are converted into text messages and then the text is converted into voice notes. All the processing is done by using MATLAB.
- In 2017, Govardhan. S.D et al. [15], developed and implemented a smart assistant system for the visually impaired. This system is a combination of a smart object detecting wearable module and a smartphone, which provides critical environmental information for the visually impaired to navigate their surroundings. This system uses the Arduino and two sensors, an ultrasonic sensor and a motion sensor. The ultrasonic sensor is used to recognize obstacles and the motion sensor is used to determine living things. Finally, the proposed system is able to detect obstacles and living things by the voice recognition system.
- In 2018, Nishajith.A et al. [16], developed and built a smart cap to aid visually impaired persons in navigating freely by allowing them to experience their environment. A camera will be used to capture the scene around the person, and the objects in the scene will be detected. The earphones will produce a speech output that describes the items that have been detected. Raspberry Pi3 processor, camera, earphones, and power source are all parts of the system's design.

The device makes use of the TensorFlow (TF) API, which is an open-source machine learning framework for object detection and classification developed by the Google Brain team. The details of the detected item are converted to speech using text to speech synthesizer (TTS) software.

- In 2018, M. Ghilardi et al. [7], investigated the current state-of-the-art in localization, detection and classification based on deep neural networks, and presented a solution for the detection of pedestrian traffic lights together with their current state for helping visually impaired people to cross the streets with the aid of their mobile device. The authors provided a novel public dataset with 4,399 labeled images of pedestrian traffic lights and presented a detailed comparison among the state-of-the-art methods for classification, localization and detection.
- In 2019, S. Shadi et al. [17], developed a mobile-based navigation system for helping visually impaired people with outdoor navigation. The proposed system will be able to reduce the obstacle collision risks by enabling users to walk outside smoothly with voice awareness. The suggested solution includes a mobile-based camera and deep learning algorithms which are employed for recognizing and detecting different objects. The system is used to help visually impaired people to navigate in unfamiliar environments. The suggested smartphone-based system is not restricted to the defined outdoor environments and does not depend on any other positioning systems.

- In 2019, B. Kaur and J. Bhattacharya [18], represented a cost-effective scene perception system aimed towards visually impaired people. An android system integrated with a USB camera and USB laser that is attached to the chest. The object detection and classification framework exploits a multi-modal fusion based on faster region based convolutional neural networks (RCNN) using motion, sharpening, and blurring filters for efficient feature representation.
- In 2019, R. Ribani and M. Marengoni [19], proposed a system that implements sensory substitution of vision through a wearable item with vibration motors positioned on the back of the user. The system used deep learning techniques to detect and classify objects in controlled environments. The hardware consists of a simple HD camera, two Arduinos, 9 cylindrical DC motors, and a Raspberry Pi. A single shot detector (SSD) with ResNet-50 performs a real-time detection implemented on the Raspberry Pi, sending the detected object class and location as defined patterns to the motors.
- In 2019, A. Bhandari et al. [20], proposed a system that would provide a low-cost portable solution to assist blind people. It consists of shoes having ultrasonic sensors to survey the scene. Once the sensors detect the object in front of the person, the camera module gets activated and captures the image, and then it will be processed to detect the object type, according to which information will be sent as audio via a Bluetooth headset to the user. The visually impaired will receive audio instructions about the obstacles in the dynamic environment.

- In 2020, M. Afif et al. [21], proposed a new indoor object detection system. The framework is built based on the deep convolutional neural network. Evaluation is done by using various backbones such as residual neural network (Resent) and dense convolutional network (DenseNet) to improve detection performance and processing time.
- In 2020, A. Salem [22], presented an enhancement and development of smart glass system for visually impaired people by using an intelligent system to improve the quality of their lives. For totally blind people, there are auditory cues developed to inform the direction where they can go ahead, an intelligent system to recognize the objects that can face visually impaired people through their mobility in an indoor environment or travelling. Combining a support vector machine and a Gray Wolfe Optimizer yield a powerful hybrid intelligent method for object classification.
- In 2020, K. Singh [23], proposed a system to detect Indian paper currencies, which includes six kinds of currency banknote. The proposed system takes a given image as input and pre-processes it. The inner and outer edges are then extracted using a Sobel algorithm. Clustering will be done using You Only Look Once (YOLO) algorithm. After that, the recognized input image is compared with the features of the image and classified as 200, 500, 2000, or not with the help of the YOLOv3 algorithm.
- In 2020, L. Abraham et al. [8], proposed a system for identifying walkable spaces, text recognition and text to speech, identifying and locating specific types of objects, and walking navigation. This is implemented with the YOLO algorithm for object detection, which uses the Common Objects in Context (COCO) dataset.

This system helps a blind person to walk easily by finding the path, detecting obstacles and thus avoiding them.

- In 2020, S. Shaikh [24], presented a method to assist the visually impaired by developing a system that captures real-time pictures in front of the users using a USB camera. For feature extraction, the YOLO machine learning approach is utilized. This system uses text-to-speech technology to provide audio descriptions of the environment, allowing visually impaired people to move safely.

1.3 Problem Definition

The problem of the study can be modeled using the following four questions:

1. What are the most common objects that a dataset should include, and how are they collected properly? How much training data is needed to achieve a good performance?
2. How existing deep learning models could be leveraged to perform object detection and classification with the goal of assisting visually impaired people?
3. What is the accuracy of the trained deep learning model? How to create a deep learning model with high accuracy, speed, and robustness for object detection at a low cost to assist visually impaired people?
4. How to propose a system accessible to visually impaired people?

All these questions exist within the context of proposing a system that will assist visually impaired people to detect objects and provide necessary information about those objects.

1.4 Aims of the Study

A synthesis of theory and experimentation will use to answer the questions outlined in section 1.3. This combination of theory and experimentation will lead to well-structured experiments with deep theoretical foundations. The experiments will test in real-world conditions, which is an important step to successful implementation. The research conducted in different stages and these stages were:

1) A custom dataset will be collected, which involves performing the pre-processing steps needed to make the image in the dataset appropriate to be used as the input of a deep learning model.
2) Implementing the YOLOv3 deep learning algorithm by using python, then performing a training process with modification on hyper parameters of the YOLOv3 deep learning algorithm to produce the final model architecture with high accuracy. Then using the trained models in the proposed system.
3) The proposed system will be implemented on a Raspberry Pi processor and compared to a NVIDIA Jetson Nano processor to produce object detection to assist visually impaired people in their daily lives.

The system's process begins with image acquisition, which is done by using a camera. The camera captured real-time frames, then sets them to the deep learning trained model. The system must designed to provide audio output to visually impaired people by using text to speech (TTS) toolkit.

1.5 Thesis Layout

Chapter two provides a general introduction to computer vision and object detection. The general architecture of convolutional neural networks (CNN) is also included. As well as details related to the YOLO algorithm. Finally, the last section illustrates the hardware equipment that is used in model training.

Chapter four presents the training and accuracy results obtained before and after applying the optimization, and the analyzing of these results in order to develop the proposed system and achieve the desired accuracy.

Chapter five presents the hardware implementation of the proposed system on a Raspberry Pi processor and a NVIDIA Jetson Nano processor to produce an object detection system to assist visually impaired people in their daily lives.

Chapter six presents conclusions and suggests trends for future work.

CHAPTER TWO

Computer Vision and Object Detection

2.1 Introduction

This chapter provides a general introduction to computer vision and objects detection. The general architecture of CNN is also included. Moreover, details related to the YOLO algorithm, like features extraction, challenges, versions and some additional considerations which are relevant to the design of the deep neural network are presented in this thesis. Finally, the last section illustrates the hardware equipment that is used in model training.

2.2 Object Detection

Artificial Intelligence (AI) has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of computer vision.

AI is a branch of Computer vision that allows machines and systems to extract useful information from digital pictures, videos, and other visual inputs and take action or create recommendations based on that data. If AI allows machines to think, computer vision allows them to see, observe, and understand [25]. One of the most important and challenging branches of computer vision is object detection [26].

Object detection can be identified as a technology that mimics the functions of human visual systems to detect objects in images and videos and the task of localizing and classifying objects in a given image or video.

Object detection is closely linked to other computer vision technologies like image segmentation and image recognition since it helps us comprehend and analyze situations in images or videos. However, there are substantial differences. Image segmentation understands the elements of a scene at the pixel level, whereas image recognition only outputs a class label for an identified object. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. Object detection relates to many applications such as assisted and autonomous driving, navigation aids for the visually impaired, robotics applications, and landmark detection etc. [27] [28]. Traditional machine learning-based techniques and deep learning-based approaches are two types of object detection algorithms [9].

2.2.1 Object Detection Using Machine Learning-Based Approaches

Computer vision techniques are used with more traditional machine learning-based approaches to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the object's location as well as its label. So, the pipeline of traditional object detection models can be divided into three stages [29]:

1. **Feature Extraction:** To detect different objects, we need to extract visual features that can provide a semantic and robust representation. Scale-invariant feature transform (SIFT) [30], histograms of oriented gradients (HOG) [31], and Haar-like [32] features are the representative ones.

This is because these features can create representations associated with complex cells in the human brain. However, due to the diversity of appearances, illumination conditions and backgrounds, it's difficult to manually design a robust feature descriptor to perfectly describe all kinds of objects [29].

2. **Informative Region Selection:** Objects may appear in any position in the image and have different aspect ratios. Scanning the full image with a multi-scale sliding window is a normal choice. Although this robust strategy can detect all object locations, its shortcomings are also obvious. It is computationally costly because of the many candidate windows, and it creates too many redundant windows. However, if only a fixed number of sliding window templates is applied, unsatisfactory regions may be produced [29].
3. **Classification:** A classifier is needed to distinguish a target object from all other categories and make the representations more hierarchical, semantic and informative for visual recognition. Usually, the supported vector machine (SVM) [33], AdaBoost [34], and the deformable part-based model (DPM) [35] are good choices.

Small gains are obtained using this three-step cascated. This is correct for the following reasons. The first reason is that the generation of candidate bounding boxes with a sliding window strategy is redundant, inefficient and inaccurate.

The second reason is that the semantic gap cannot be bridged by the combination of manually engineered low-level descriptors and discriminatively-trained shallow models.

2.2.2 Object Detection Using Deep Learning-Based Approaches

Object detection has gained a lot of attention in recent years because of its wide range of applications and recent technical breakthroughs. Among the many reasons and efforts that have led to the fast evolution of object detection techniques, notable contributions should be attributed to the fast progression of deep learning and the continuous improvement of computing power [26].

Deep learning algorithms have emerged as the state-of-the-art approach to object detection. Deep learning-based object identification models are typically divided into two parts. An encoder takes an image as input and processes it via a series of blocks and layers that learn to extract statistical characteristics that may be used to find and label objects. The encoder's output is then sent to a decoder, which predicts the bounding boxes and labels for each object.

When the first convolutional neural network-based object detector was proposed, a series of significant contributions were made which promoted the development of general object detection by a large margin to address the problems existing in traditional machine learning-based approaches [26] [36].

2.3 Convolutional Neural Network (CNN)

CNN is the most representative deep learning model. It consists of a multilayered neural network with a special architecture to detect complex features in data. Each CNN layer is referred to as a feature map. The input layer's feature map is a 3D matrix of pixel intensities for distinct color channels. Any internal layer's feature map is an induced multi-channel image, whose ‘pixel’ can be viewed as a specific feature. Every neuron is linked to a limited number of neurons from the previous layer[29]. CNN is designed to learn spatial hierarchies of features automatically and adaptive through back propagation by utilizing multiple building blocks such as convolutional layers (conv layers), fully connected layers (Fc layers), pooling layers, and activation functions such as rectified linear unit (ReLU). CNN has been used in image recognition, powering vision in robots, and for self-driving vehicles [37] [38] [39]. Figure 2.1 shows an example of CNN.

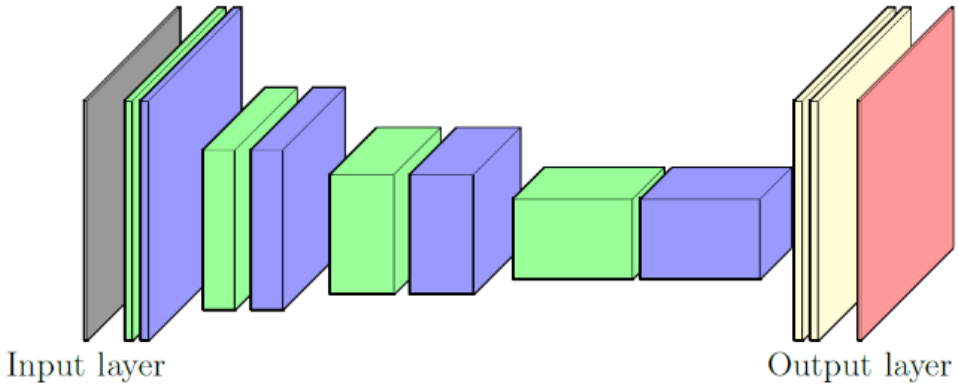


Figure 2.1 Example of CNN. Green layers are convolutional layers, blue are pooling layers and yellow are fully connected layers. Gray is the input layer and red is the output layer [40].

CNN can be created and trained to solve image classification and object detection. In image classification, the goal is to identify object in an image. In object detection, the goal is to identify and also locate the objects in an image [41].

2.3.1 Convolutional Layer

Convolution layers are the primary building blocks of a CNN. A convolution layer consists of a set of learnable 2D filters. Usually, each filter has a small spatial extent (width and height), but it extends through the full depth of the input feature. The layer executes a convolutional process on the input and filters before passing the output to the next layer. The convolution produces a 2D plane known as a feature map. Typically, a large number of learnable filters are used to generate the output of the convolutional layer in three dimensions, $H \times W \times D$ where H and W are the spatial dimensions and D is the number of feature maps (numbers of filters used) [42] [29].

Equation 2.1 is a two-dimensional discrete convolution with steps from (0, 0) to(m, n) [41]:

$$S(i,j) = \sum_m \sum_n \mathrm{K(m,n)I(i-m, j-n)} \quad \text{.....} \quad (2.1)$$

Where:

S: represents the output.

I: represents the input image.

K: represents the kernel.

i, j : represents coordinates in the output feature map S.

$(\mathrm{i-m, j-n})$ Represents the coordinates in input I.

$\mathrm{m, n}$: represents the coordinates in the kernel K.

To maintain the spatial dimensions, the input volume is often padded with zeros. The spatial dimension of the output is determined by Equation (2.2)

$$W_{out} = \frac{Win-K+2P}{S} + 1 \qquad \text{.....} \qquad [41]\ (2.2)$$

Where:

W_{out}: Output spatial dimension.

W_{in}: Input spatial dimension.

K: Kernel size.

S: stride.

P: zero padding.

A convolutional layer with three 3×3 filters, stride $S = 1$, and padding size $P = 1$ is shown in figure 2.2.

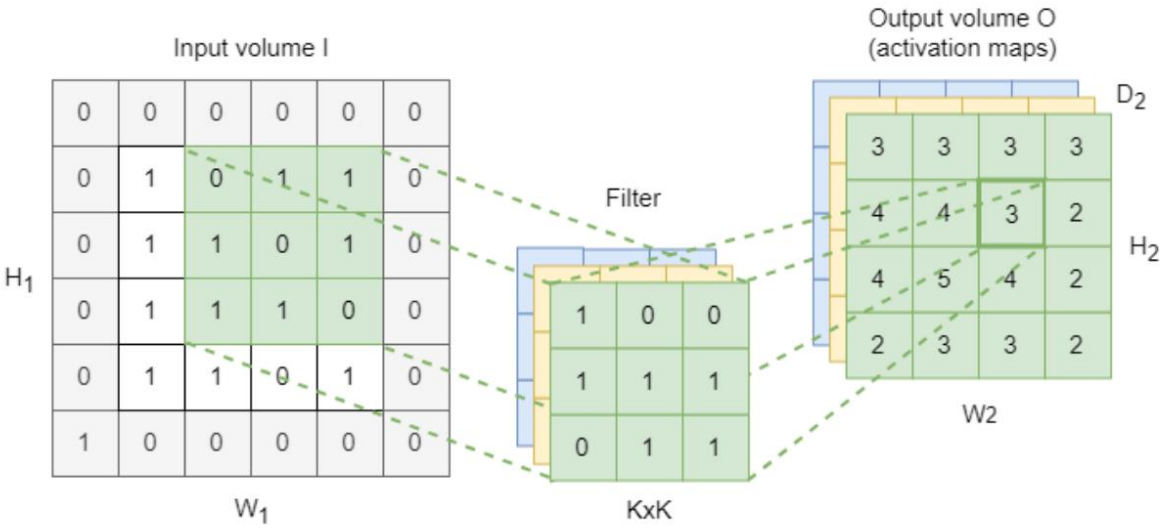


Figure 2.2 Convolutional layer [43]

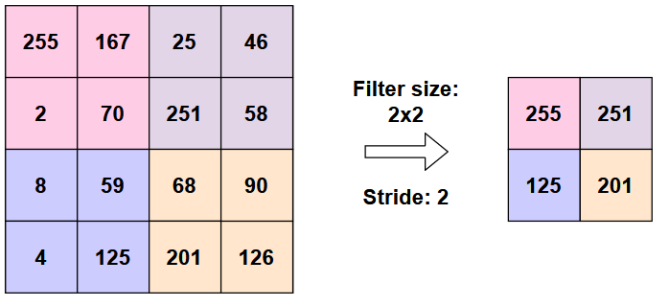
2.3.2 Pooling Layer

The pooling layer, like the convolutional layer, is responsible for decreasing the spatial size of the convolved feature. By decreasing the dimensionality of the data, the processing power required to process it is reduced. Furthermore, it is beneficial for extracting dominating features that are rotational and positional invariant, allowing the model to be efficiently trained [43].

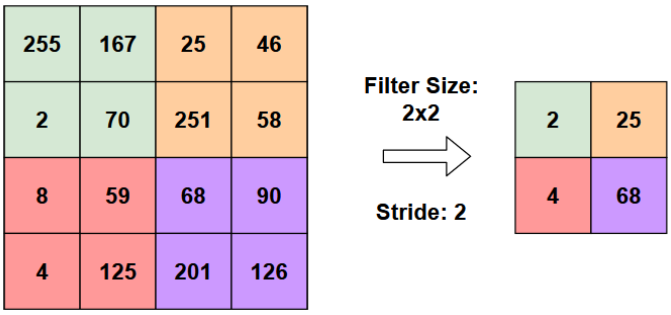
Pooling is classified into three types: MAX pooling, MIN pooling and average pooling. Max Pooling provides the maximum value from the kernel-covered region of the image, which is mostly used when the image has a dark background since max pooling will select brighter pixels. In MIN pooling, the summary of the features in a region is represented by the minimum value in that region. It is mostly used when the image has a light background, since min pooling will select darker pixels. Average Pooling, on the other hand, provides the average of all the values from the region of the image covered by the kernel, which smooths the harsh edges of a picture and is used when such edges are not important [41].

Max Pooling can also be used to reduce noise. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

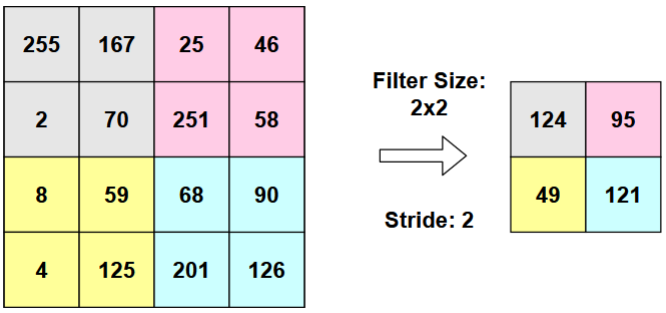
On the other hand, average pooling simply performs dimensionality reduction as a noise suppressing mechanism. As a result, we can conclude that Max Pooling performs better than average pooling for this application. Figure 2.3 shows the operation of the pooling layer.



MAX pooling



MIN pooling



Average pooling

Figure 2.3 Pooling layer with kernel size 2×2 and stride 2

The convolutional layer and the pooling layer, together, form the i^{-th} layer of a convolutional neural network. Depending on the complexity of the images, the number of such layers can be increased to capture even more low-level details, but at the expense of more computational power.

2.3.3 Rectified Linear Unit Layer (ReLU)

The ReLU is an activation function that is applied to the feature map, resulting from the convolution of the filters with the input array. As seen in Figure 2.4, the ReLU is a non-linear mechanism that is used instead of tangent and sigmoid activation functions. It helps to prevent the vanishing gradient problem, while maintaining the positive features and speeding up deep network training [38].

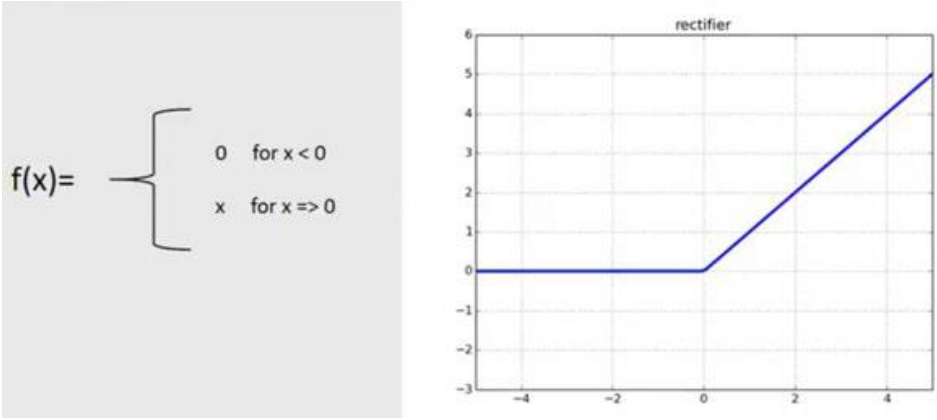


Figure 2.4 ReLU activation function [23]

2.3.4 Batch Normalization

Deep Neural Network training is complicated by the fact that the distribution of each layer's inputs varies during training as the parameters of the previous layers change. This slows down training by requiring lower learning rates and more careful parameter initialization. This phenomenon is known as an internal covariate shift. Batch normalization is a process to make neural networks faster and solve the internal covariate shift problem through adding extra layers to a deep neural network. The new layer performs a normalization operation on the input of a layer coming from the previous layer [44].

2.3.5 Fully Connected Layer (FC)

Fully connected layers connect the activation in the input feature map to each activation in the output feature map. Typically, these activations are calculated using matrix multiplication. Modern CNN architecture uses FC layers for classification. The output of the convolutional and pooling layers represents high-level features.

Most of these features may be good for classification, but their combination might be even better. Therefore, the last layer of a CNN is the FC layer, as it is usually a cheap way of learning non-linear combinations of its input features. The last FC layer for classifiers uses SoftMax as the activation function, as it converts an input vector to probabilities summing to one [41] [43].

The main distinction between the FC and the convolutional layers is the neurons in the convolutional layer are only connected to a local area of the input, and many of them share parameters. Besides that, both layers compute dot products, and hence, any FC layer can be rewritten as a convolutional layer and vice versa.

2.4 You Only Look Once (YOLO)

When humans look at a picture, they immediately recognize what objects are present, where they are located, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate object detection algorithms will allow computers to drive cars without specialized sensors, assistive devices to relay real-time scenario details to human users, and unlock the potential for general-purpose, responsive robotic systems.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image.

For object detection problems, CNNs are too slow and computationally costly. CNNs could not be trained on too many patches created by the sliding window detector. The region based convolutional neural network (R-CNN) [36] solves this problem by using a selective search object proposal algorithm, which decreases the number of bounding boxes fed to the classifier to about 2000 region proposals. Running CNN on 2000 regional proposals created by a selective search, on the other hand, takes a long time.

Girshick et al. wrote a second paper, Fast R-CNN, in 2015 [45]. The Fast R-CNN algorithm outperformed the original R-CNN algorithm in terms of precision and time required to complete a forward pass. However, the model also depends on an external region proposal algorithm. R-CNN and Fast R-CNN are also used selective searches to find the region's proposal, which made them both slow. Shaoqing Renetal. [46] has developed an object detection algorithm that replaces the selective search algorithm and allows the network to learn the region proposals.

All the methods mentioned so far examine the image multiple times, which increases the total computational time. Both the single shot detector (SSD) [47] and YOLO [48] use a one-stage detector technique to further improve the speed of deep learning-based object detectors, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities. SSD eliminates proposal generation and is simpler to train. The accuracy of SSD is close to that of Faster-RCNN and its speed is close to the YOLO algorithm.

In 2015, Redmon et al. [49] proposed the regression-based YOLO algorithm. Instead of proposing regions and classifying those regions, the YOLO algorithm, as its name suggests, looks at the complete image and uses a single network layer to predict the bounding boxes and their confidence levels. It is the state-of-the-art technology used for object detection [49].

It's worth noting that YOLO just runs the image via CNN once. As a result, images can be processed in real time. The original YOLO algorithm is exceptionally fast at 45 FPS, and is twice as accurate as other real-time algorithms of that time. However, it still lags behind in accuracy compared with R-CNN.

YOLOv2 [50] is the second version of YOLO that improves accuracy significantly while making it faster. YOLOv2 is concentrated on improving recall and localization. To enhance YOLO's results, a variety of previous ideas and new concepts are combined. The most important of those is the introduction of anchor boxes, as proposed in the Faster R-CNN algorithm. Using anchor boxes instead of directly calculating bounding box coordinates simplifies the issue and allows the network to learn more easily.

YOLOv2 works with 416×416 pixel input images, and its convolutional and pooling layers down-sample the image by a factor of 32 to produce a feature map with a size of 13×13. YOLOv2 has the extra advantage that it has a single center cell. Large objects tend to occupy the center of the image, so it helps to have a single center cell to predict these objects, instead of 4 close-by cells.

The mean average accuracy (mAP) on the VOC2007 test set grows from 67.4 percent to 78.6 percent when compared to the previous generation. However, because each cell is only responsible for predicting one item, the recognition proves insufficient when faced with the objective of overlapping.

YOLOv3 is a new version of the algorithm with incremental improvements. YOLOv3 predicts on three distinct scales. It predicts more bounding boxes per image and is also more effective at detecting tiny images. When the anchor box overlaps a ground truth box more than every other anchor box, the abjectness score is expected to be 1, which is dependent on logistic regression. Also, instead of using a SoftMax classifier to predict the classes of a predicted bounding box, they use independent logistic classifiers for each class. A SoftMax classifier is unnecessary for good performance, and a SoftMax classifier imposes the assumption that each box has exactly one class[51].

On the COCO dataset, the mAP50 increases from 44.0 percent for YOLOv2 to 57.9 percent. In comparison to RetinaNet, which has a 61.1 percent, RetinaNet has a 500 input size. The detection speed for 500∗500 is around 98 ms/frame, but YOLOv3 has a detection speed of 29 ms/frame when the input size is 416×416. Figure 2.5 shows a comparison of the mAP evaluation metric for object detection algorithms.

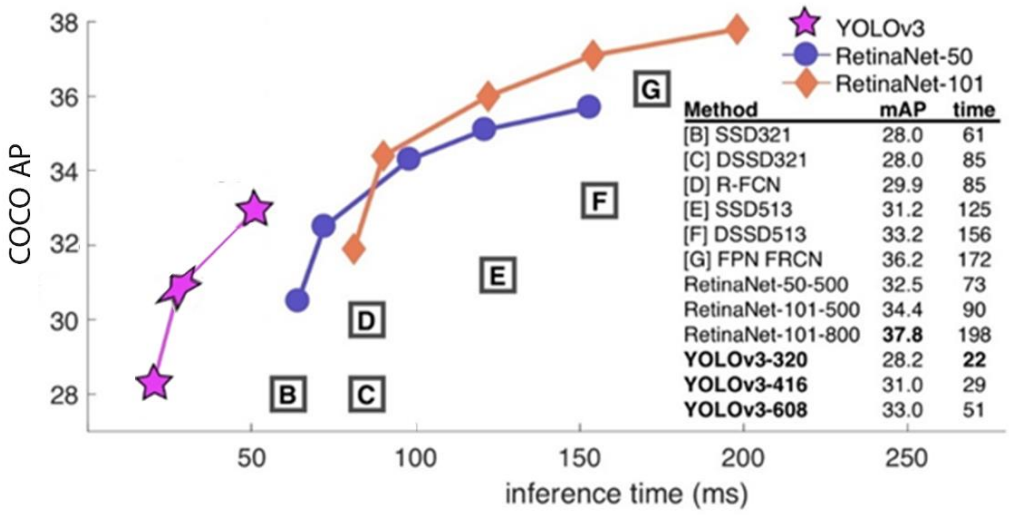


Figure 2.5 mAP comparison of object detection algorithms

YOLOv3 predicts the bounding boxes and their class probabilities using a single forward pass neural network deployed to the whole image. This method allows YOLOv3 to be quite fast without losing a lot of accuracy. YOLOv3 has 53 convolutional layers called Darknet-53, which are mainly composed of Convolutional and Residual structures.

YOLOv3 predicts on three different scales. The detection layer is used to make predictions on feature maps with strides of 32, 16, and 8, respectively. The network down-samples the input image until the first detection layer, where a detection is made using feature maps of a layer with stride 32. Further, layers are up-sampled by a factor of 2 and concatenated with feature maps of previous layers having identical feature map sizes. Another detection is now made in the layer of stride 16. The same up-sampling procedure is repeated, and a final detection is made in the layer of stride 8.

2.5 Residual Network (ResNet)

A single-layer feedforward network is considered to be sufficient for representing any function. The layer, on the other hand, could be very large, and the network could be prone to overfitting the data. Therefore, in the research community, there is a common trend that network architecture could go deeper.

An increase in network depth by simply stacking layers together is insufficient. Because of the vanishing gradient problem, deep neural networks are difficult to train (by making the CNN deeper, the derivative becomes almost insignificant in value when back-propagating to the initial layers). As a consequence, as the network becomes deeper, its performance becomes saturated or rapidly decreases, as shown in figure 2.6.

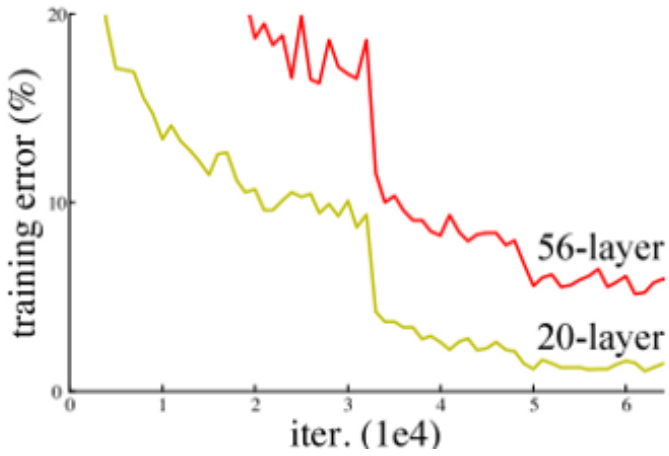


Figure 2.6 Increment of network depth leading to worse performance [53]

There are various methods for dealing with the vanishing gradient problem prior to ResNet, such as inserting an auxiliary loss in the middle layer as extra supervision. However, none of them seemed to address the problem directly. ResNet's key idea is to introduce "identity shortcut connections" that skip one or more layers, as shown in figure 2.7.

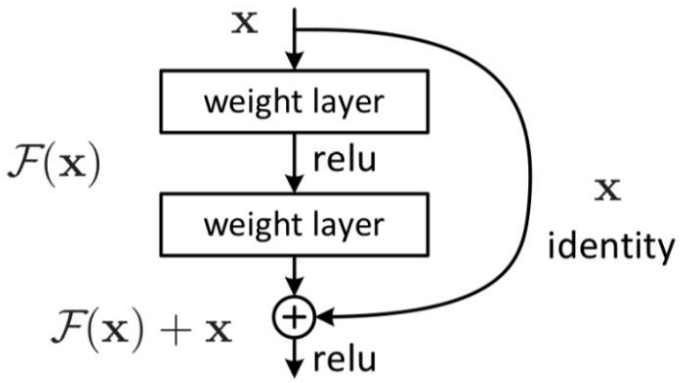


Figure 2.7 Basic building block of ResNet [53]

The idea is to reroute the input and add the learned concepts from the previous network. According to the principle, the next layer will learn from both the contributions of the previous layer's input and from the previous layer itself. In comparison to other networks, it is intended to improve the training process [53].

2.6 Transfer Learning

In many real world applications, it is expensive or impossible to recollect the needed training data and rebuild the models. It would be nice to reduce the need and effort to recollect the training data. In such cases, knowledge transfer or transfer learning between task domains would be desirable [54].

Transfer learning is a commonly used design method to mitigate the effects of too few training images or the lack of processing power. It works by reusing the weight configuration from a pre-trained network on a new network.

Traditional machine learning techniques try to learn each task from scratch, while transfer learning techniques try to transfer the knowledge from some previous tasks to a target task when the latter has less training data [54] [55]. Transfer learning works because the early layers of a CNN typically learn basic features such as edges and corners, which many objects have in common. The model in the target domain does not need to be trained from scratch, which can significantly reduce the demand for training data and training time in the target domain [56].

There are two types of transfer learning. With frozen layers, the imported weights do not change during further training, and with fine-tuning, the imported weights are further trained [41].

2.7 Object Detection Evaluation Metrics

Object detection evaluation metrics serve as a measure to assess how well the model performs on an object detection task. It also enables users to compare multiple detection systems objectively or compare them to a benchmark.

Since the classification task only evaluates the probability of the class object appearing in the image, it is a straightforward task for a classifier to identify correct predictions from incorrect ones. However, the object detection task localizes the object further with a bounding box associated with its corresponding confidence score to report how certain the bounding box of the object class is detected.

2.7.1 Intersection over Union (IoU)

The IoU, also known as the Jaccard index [57], is the most common metric used for comparing the similarity between two arbitrary shapes. In the context of object detection, the IoU is defined as the overlapping area between the predicted and real bounding boxes divided by the area of union between them [58], which is attained as [58] :

$$J(B_p, B_{gt}) = IOU = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \quad \text{.....} \quad (2.3)$$

Where:

B_p : predicted bounding box.

B_{gt} : Ground-truth bounding box.

Figure 2.8 depicts the IoU.

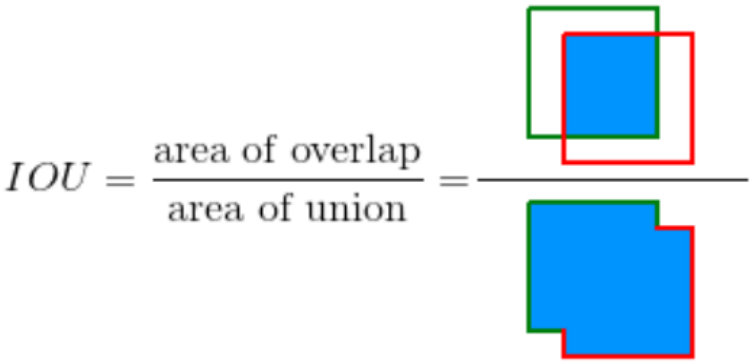


Figure 2.8 IoU

When the IoU is compared to a specified threshold t, the detection is considered accurate if $IoU \geq t$ and the detection is considered incorrect if $IoU < t$ [58].

2.7.2 Generalized Intersection over Union (GIoU)

IoU is commonly used in academia to evaluate the similarity of two bounding boxes. However, IoU has major disadvantages that make it unsuitable for the loss function. When there is no coincidence between the prediction box and the real box, the first disadvantage occurs. The IoU value is zero, resulting in a gradient of zero when optimizing the loss function, indicating that it cannot be optimized. In figure 2.9, Scene A and Scene B both have an IoU value of zero, but Scene B has a better prediction impact than Scene A because the distance between the two bounding boxes is less.

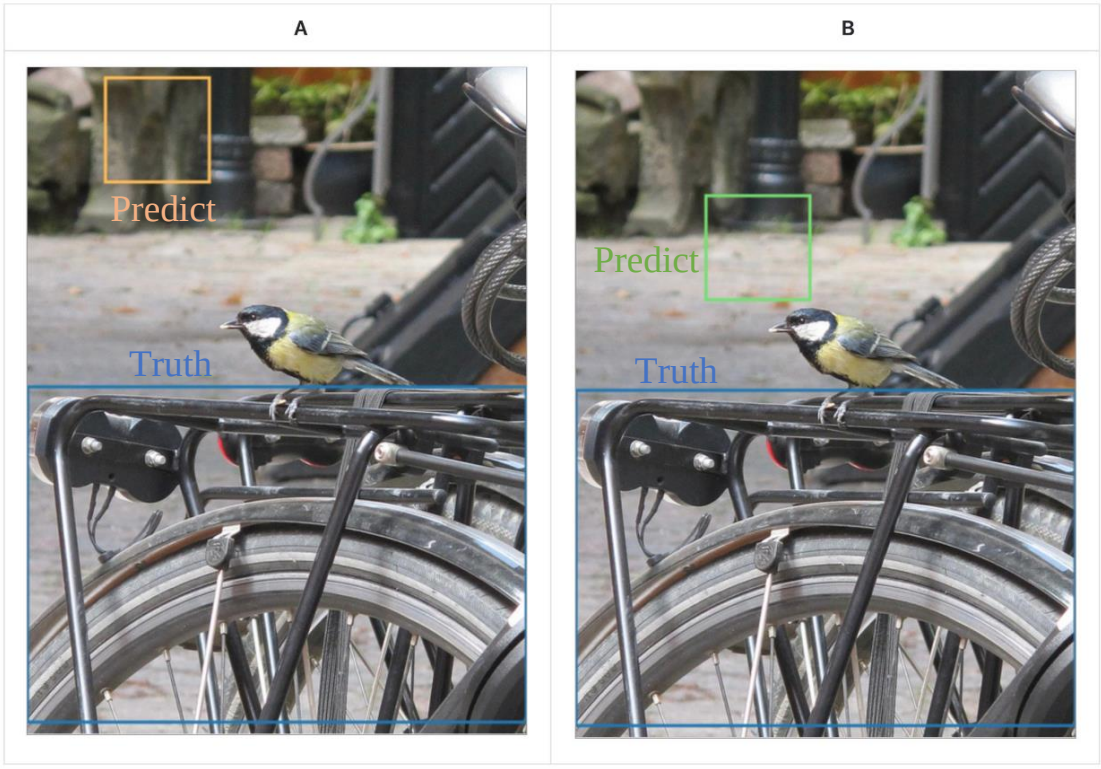


Figure 2.5 Scene A and scene B

When the prediction box and the real box overlap and have the same IoU value, the second disadvantage occurs. As seen in figure 2.10, the detection impact has a large difference.

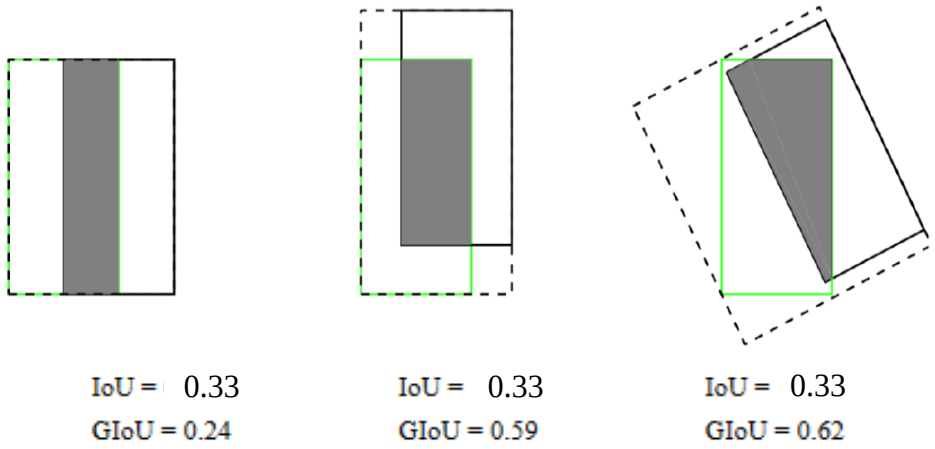


Figure 2.60 IoU between three boxes

To address these issues, H. Rezatofighi et al. [57] proposes the generalized version of IoU as a new metric for comparing any two arbitrary shapes. By incorporating GIoU loss into state-of-the-art object detection algorithms, this consistently improves their performance on popular object detection benchmarks.

The IoU for determining the similarity between two arbitrary shapes $A, B \subseteq S \in R^n$ is attained as [59]:

$$IoU = \frac{|A \cap B|}{|A \cup B|} \quad \text{.....} \quad (2.4)$$

For A and B, find the smallest enclosing convex object $C, where\ C \subseteq S \in R^n$. Where, A is the output result, B is the Ground Truth, S is overlapping space.

The equation of GIoU as follows [59] :

$$\text{GIoU= IoU-} \frac{|C/(A \cup B)|}{|C|} \quad \text{.....} \quad (2.5)$$

2.7.3 Average precision (AP)

AP is a popular metric for measuring the detection accuracy of image and video object detectors involves calculating the area under the curve (AUC) of the relationship between recall (R) and precision (P) [58].

Precision is a model's ability to recognize only relevant objects. It is the percentage of correct positive predictions, whereas recall is a model's capacity to discover all relevant cases. It is the percentage of correct positive predictions made out of all provided ground truths. [58]. Precision and recall are defined by the following equation:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{TP}{all\ detections} \quad \text{.....} \quad [58]\ (2.6)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{TP}{all\ \text{ground truths}} \quad \text{.....} \quad [58]\ (2.7)$$

Where:

TP: True Positives.

FP: False Positives.

FN: False Negatives.

For object detection, the concept of IoU is used to measure how much the predicted boundary overlaps with the ground truth. The threshold value for IoU is used to determine if the object detection is valid or not. Set the IoU to 0.5, and in that case:

- If the IoU is greater than 0.5, the object detection is classified as a TP.
- If the IoU is less than 0.5, the detection is incorrect and is classified as a FP.
- When the ground truth is present in the image but the model fails to detect it, it is classified as a FN.

- True Negative (TN) is present when the model does not predict an object. Because this metric is ineffective for object detection, it is ignored.

The mAP [60] is a metric for evaluating object detector accuracy across all classes in a database. The mAP is the average AP across all classes, is[58]:

$$\text{mAP}= \frac{1}{N}\sum_{i=1}^{N} AP_i \qquad \text{.....} \qquad (2.8)$$

Where

N: is the total number of classes that are being evaluated.

AP_i : is the AP in the ith class.

2.8 Non-Maximum Suppression (NMS)

NMS is a strategy used in many computer vision algorithms. It is a type of algorithms to select one entity, such as bounding boxes, out of many overlapping entities. The criteria can be chosen to provide specific results. Most typically, the requirements are some kind of probability number and some kind of overlap metric, such as IoU [60] [61].

Most object detection algorithms utilize NMS to reduce a large number of observed rectangles to a few. The use of NMS is necessitated by the way most object detection algorithms, such as the Faster RCNN, YOLOv3, and SSD.

Most object detectors do some form of windowing at the most basic level. Hundreds of thousands of windows of various sizes and shapes are produced, either directly on the picture or on an image feature.

These windows are supposedly to have only one object, and a classifier is used to calculate a probability score for each class. When the detector generates a large number of bounding boxes, the best ones must be chosen. The most widely used algorithm for this task is NMS. To eliminate duplicate detections, NMS is utilized as a post-processing step.

2.9 Anchor Boxes

Essentially, a grid cell can detect only one object whose mid-point of the object falls inside the cell. However, if a grid cell includes more than one mid-point of the object, this indicates that multiple objects are overlapping. Anchor boxes are used to alleviate this issue. Anchor boxes are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes that models want to detect and are typically chosen based on object sizes in training datasets [50].

The typical task of training an object detection network consists of proposing anchor boxes or searching for potential anchors, then pairing proposed anchors with possible ground truth boxes. It is important to note that the concept of anchor boxes can be applied to predict a fixed number of boxes. The use of anchor boxes enables a network to detect multiple objects, objects of different scales, and overlapping objects [62].

2.10 Graphics Processing Unit (GPU)

The performance of deep learning approaches increases with respect to the increment in the amount of data, as demonstrated in figure 2.11. Moreover, big datasets have become widely freely available for researchers for many purposes. Thus, deep learning needs a significant amount of computer power in order to run well. The training phase of a deep learning model is the most resource-intensive task for any neural network [39].

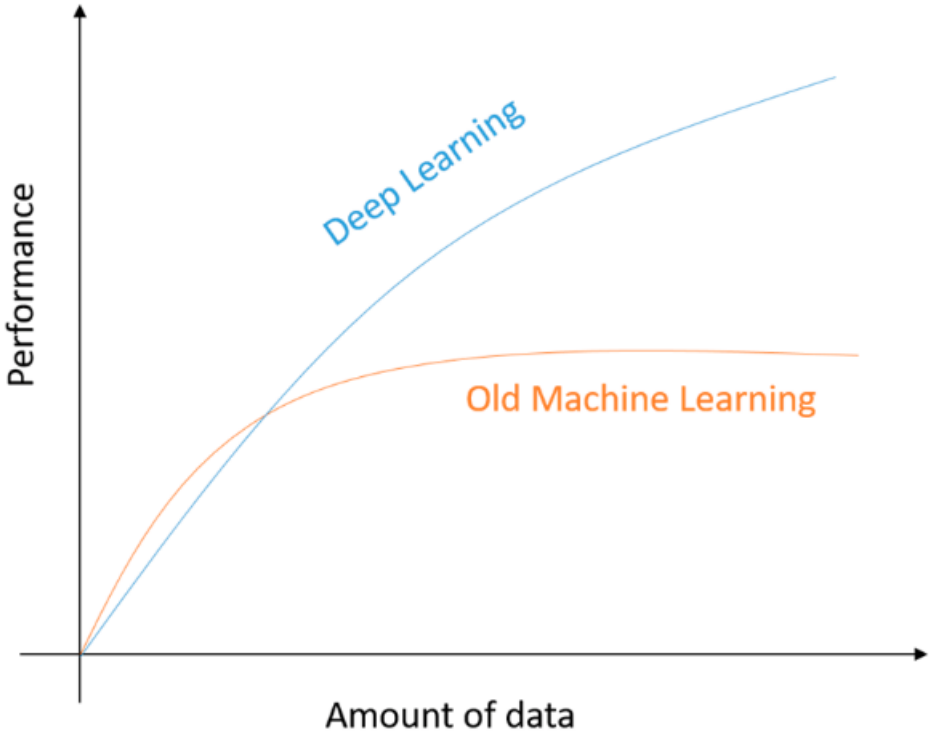


Figure 2.7 Performance of deep learning with respect to the amount of data [39]

In a couple of minutes or hours, a PC would be able to manage a neural network with 10, 100, or 100,000 parameters. However, training a neural network with more than 10 billion parameters would take years using the old technique.

A GPU is an advanced processor with dedicated memory that conducts the floating point calculations required for graphics rendering. It is, in other words, a single-chip processor that performs complex graphics and numerical computations while freeing up CPU resources for all other tasks. The CPU serves as the host, and the GPU serves as a device. Although a GPU is smaller than a CPU, it has more logical cores.

Interest in GPU computing blossomed when this potential was combined with a programming language that made GPUs easier to program. Therefore, many programmers of scientific and multimedia applications today are thinking about whether to use GPUs or CPUs. For programmers interested in deep learning, GPUs are currently the preferred platform [64].

2.10.1 Programming the GPU

The early GPUs are built as graphics accelerators, but as the 1990s progressed, they became more programmable, culminating in NVIDIA's first GPU in 1999. Researchers and scientists are rapidly beginning to apply the excellent floating point performance of this GPU to general purpose computing.

NVIDIA launched the compute unified device architecture (CUDA) in 2006, the world's first solution for general-computing on GPUs. CUDA produces C/C++ for the system processor (host) and a C and C++ language for the GPU. The compiler and the hardware can gang thousands of CUDA threads together to utilize the various styles of parallelism within a GPU [64].

2.10.2 HW/SW GPU Compatibility

GPUs are ideal for training artificial intelligence and deep learning models because they can do many calculations at the same time. To make deep learning work on a GPU, hardware and software must be suitable for working on PCs. The NVIDIA deep learning software development kit (SDK) accelerates widely-used deep learning frameworks such as TF, Theano and Torch as well as many other deep learning applications.

The term "hardware" refers to the type of graphics card. Each graphic card has a specific microarchitecture called Gencode, and it must be CUDA enabled. Software means the type of operating system, utility program, CUDA version, and the deep learning framework type and version. Matching must be done between the graphic card Gencode and its software requirements, for various NVIDIA architectures [64].

CHAPTER THREE

Dataset Preparation and System Development

3.1 Introduction

The overall workflow of this thesis will be discussed in this chapter, which explains the hardware and software requirements, then dataset construction and preprocessing. Moreover, the structure of a proposed system which uses the YOLOv3 deep learning neural network and how it is implemented in the TF deep learning framework, then goes over the model training and the YOLOv3's loss function. Finally, the modification is applied to the hyper-parameter to improve the mAP and learning time of the model.

3.2 Hardware and Software Requirements

To prepare a computer to be a good environment for implementing and training a deep learning neural network for objects detection, NVIDIA GPU card with CUDA Compute Capability is required. The GeForce GTX 1650 GPU is used with these properties: (Compute Capability: 7.5, Core Clock: 1.56 GHz, Core Count: 16, Device Memory Size: 4.00 GB, Device Memory Bandwidth: 119.24 GB/s) on the Windows.

Additionally, the following programs, packages and libraries are installed on the computer:

1. Python version 3.7 (64-bit installer for a 64-bit machine): a high-level programming language.
2. Microsoft Visual C++ Redistributable 2019: Microsoft visual C++ runtime libraries.

3. Microsoft Visual Studio 2019: An integrated development environment (IDE).
4. NVIDIA CUDA Toolkit 10.1 and cudnn 7.6 SDK: A development platform for creating GPU-accelerated applications with high performance and primitive libraries.
5. TensorFlow-GPU 2.3.1: A high-performance numerical computing library used in the deep learning stable release of TF support for GPU-accelerated.
6. LabelImg Tool: A graphical image annotation tool.
7. yolov3.weights: Original weights file of YOLOv3 model trained on COCO dataset.

3.3 Dataset Construction

Dataset is a key of objects detection, and it is a significant challenge to create a dataset that could be used in developing a system to support the visually impaired that could be meaningfully integrated with their daily life routine. Therefore, items were chosen using a completely patient-centered approach.

The names of items that were selected mainly without any patient involvement are (250 Dinar, 500 Dinar, 1000 Dinar, 5000 Dinar, 10000 Dinar, 25000 Dinar, 50000 Dinar, Bed, Book, Bottle, Bus, Cabinetry, Camera, Car, Cat, Chair, Charger, Clothing, Coffee Cup, Computer keyboard, Dog, Electric Socket, Fan, Fire, Flip flops, Food, Fork, Fruit, Fire Extinguisher, Gas stove, Glasses, House, Kettle, Key, Knife, Laptop, Mouse, Medicine Tab, Mobile phone, Pen, Person, Refrigerator, Remote Control, Scissors, Shoe, Shower, Sink, Sofa Bed, Spoon, Stairs, Stick, Table, Television, Toilet Seat, Traffic light, Vegetable, Wastebasket, Watch, Water Dispenser, Weapon, Window).

A survey of 20 visually impaired or people who cared about them (see Appendix-A for the survey) has been conducted to select the items. The survey opens with a brief introduction to the topic to familiarize participants with the aims of the thesis. After that, the participants are asked to indicate their names, gender, and age. The participants are then asked to rate the items based on their importance in their daily lives, as well as what items are typically required to be detected. They are further asked to add any items that they missed in the survey.

Table 3.1 shows the results of the survey, noting that one of the important points is that the participants did not add any new items when they were asked to add them if there were any missing items in the survey. This means that the identified items met all the visually impairer's daily life requirements.

One of the dispersion measures has been used to classify the items, called "Mode". The items are rated as not important, rarely important, important, and most important, based on the high frequency of items in the survey results.

After excluding items that were rated as not important or rarely important, 44 items remained, which are highlighted in table 3. 1, 62 items are reduced to 44 items to reduce time for dataset preparation and YOLOv3 learning.

Table 3.1 Survey results

Items rated (not important)	Items rated (rarely important)	Items rated (important)	Items rated (most important)
Building	Bus	Book	250 Dinar
Camera	Dog	Bed	500 Dinar
Computer keyboard	Fork	Bottle	1000 Dinar
Fire	Food	Cat	5000 Dinar
Fire Extinguisher	Gas stove	Cabinetry	10000 Dinar
Fruit	Scissors	Fan	25000 Dinar
House	Watch	Glasses	50000 Dinar
Pen	Window	Kettle	Car
Traffic light		Knife	Chair
Vegetable		Laptop	Charger
		Mouse	Clothing
		Refrigerator	Coffee Cup
		shower	Electric Socket
		Sofa Bed	Flipflops
		Table	Key
		Water Dispenser	Medicine Tab
		Weapon	Mobile phone
			Person
			Remote Control
			Shoe
			Sink
			Spoon
			Stairs
			Stick
			Television
			Toilet Seat
			Wastebasket

There is no publicly available dataset containing all the needed items and good enough for training a deep learning-based approach model. Therefore, tedious hours have been spent to create a dataset that consists of 1000 images of each item that was selected based on the survey results. Three methods are used to construct the dataset:

1) Images from Google's open image dataset.
2) Manually labeled downloaded images.
3) Manually labeled camera images.

3.3.1 Images from Google's Open Image Dataset

Google's open Images V4 has a massive scale in several dimensions, with 30.1 million image-level labels for 19.8 million concepts, 15.4 million bounding boxes for 600 object classes. The open image dataset varies from most other datasets in three important aspects. First, all images have a creative commons attribution license and may therefore be used more easily. Second, images are gathered, starting with Flickr and continuing to images discovered elsewhere on the internet. As a result of excluding basic images that appear in search engines like Google image search, the dataset has a high proportion of interesting, complex images with many objects. Third, the images are not scraped based on a preset list of class names or tags, which results in natural class statistics and avoids the initial design bias of what should be in the dataset [63].

The items that are available for downloading in Google's open image dataset are (Bed, Bottle, Car, Cat, Coffee cup, Laptop, Mobile phone, Person, Sofa Bed, Stairs, Television, Weapon).The Open Images Dataset version 4 toolkit (OIDv4) is used to download all the selected items' images with their annotation files found in Google's open image dataset.

The flow diagram of downloading images using the OIDv4 toolkit is shown in figure 3.1.

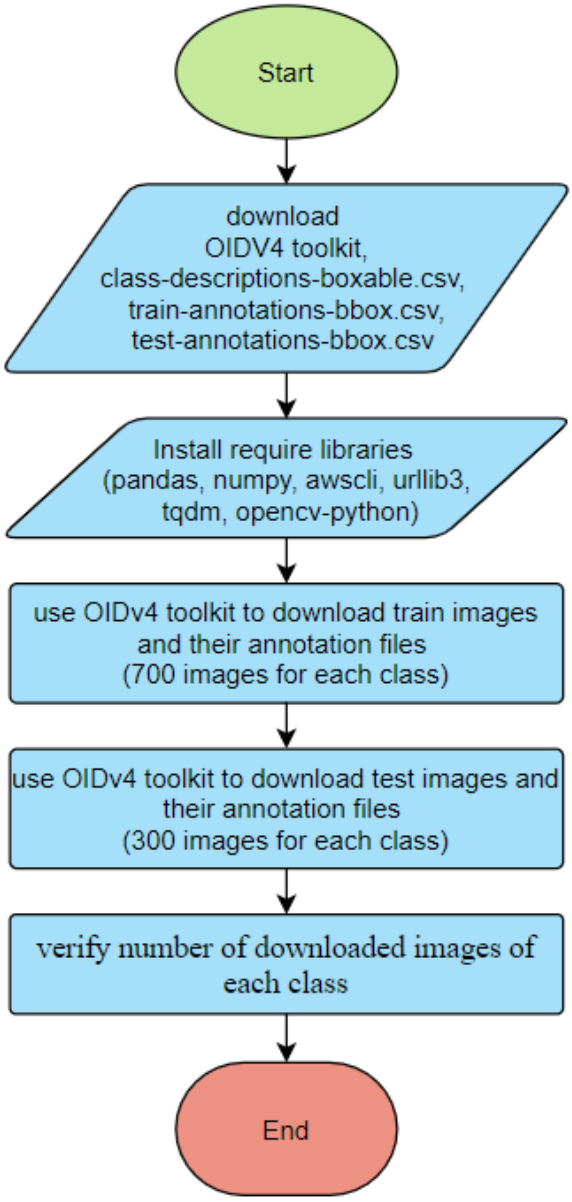


Figure 3.1 Flow diagram of downloading images using the OIDv4 toolkit.

It should be noted that the class-descriptions-boxable.csv files, as well as the train-annotations-bbox.csv and test-annotations-bbox.csv files, were required to run the OIDV4 tool kit.

The description file contains the names of all 600 classes available in Google's open image dataset, while the annotation files contain the current image's ID, bounding box coordinates, and bounding box label for the OIDv4 train and test sets.

The manual verification of image distribution in the training and test datasets led to the discovery that not all items had the required number of images. In the Google open image dataset, some items are available under multiple names. Therefore, two or three items are required to download and manually merge and change the text label files to create one class with the required image numbers. For example, "Sofa bed" is merged with "Studio couch" to create the required image number for the "Sofa bed" class, and "Weapon," "Shotgun," and "Rifle" are merged to create the required image number for the Weapon images class.

The images that are downloaded from OIDv4 are annotated in a way that is not compatible with the requirements of YOLOv3. Therefore, the dataset could not directly be trained upon.

The annotation CSV files downloaded from OIDv4 are as follows: [$\text{Class}_{\text{Name}}$ Left Top Right Bottom]. But the annotation file of the YOLOv3 model should be as following:

- One row for one image.
- Row format: [Image_File_Path Box_1 Box2 … Box_N].
- Box format: [x_{center} y_{center} width Height $Class_{id}$].

Therefore, the OIDv4 toolkit is used again in order to convert CSV files to XML files annotated in a way that is compatible with the requirements of the YOLOv3 model.

3.3.2 Manually Labeled Downloaded Images

Wearisome hours have been spent to create a dataset containing the images of these items (Book, Cabinetry, Chair, Charger, Clothing, Electric Socket, Flip flops, Fan, Glasses, Kettle, Key, Knife, Medicine Tab, Mouse, Refrigerator, Remote Control, Shoe, Shower, Sink, Spoon, Stick, Table, Toilet Seat, Wastebasket, Water Dispenser). Various internet resources were used, and nearly 25000 images were visually examined. The dataset is created with a mostly equal distribution of items after manually removing all the unsuitable images. All the samples in the dataset have varying sizes.

These items' images are labeled and bound into boxes by hand. This process is known as image labeling. Image labeling is a tedious process and very time-consuming. It takes much more time than usual. The bounding box label was inserted with great care to get precise detection.

The "LabelImg" tool is used for labeling the images and specifying where the custom objects are located on the specific image, for all dataset images, the accompanying annotation files are saved in the xml format. The labeling process is shown in figure 3.2. (For more samples, see Appendix-B)

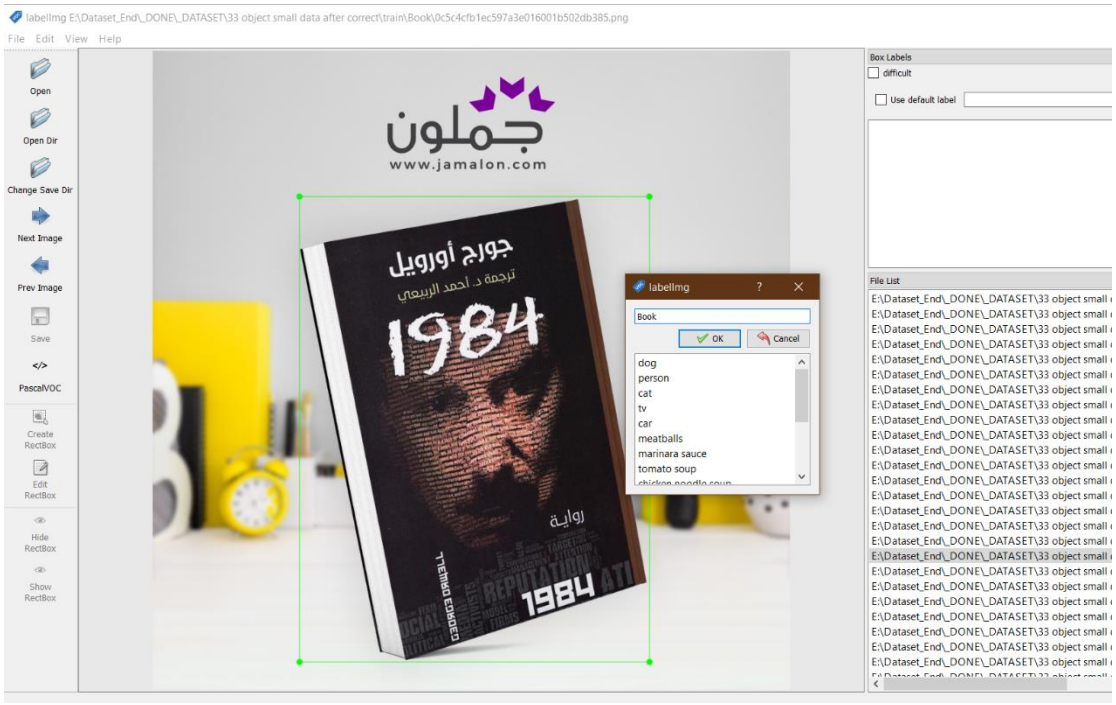


Figure 3.2 A snapshot of image labeling process

3.3.3 Manually Labeled Camera Images

The Canon 750D camera with a 24.2MP resolution is used to capture images in a variety of situations, including occlusion and illumination, and so on. Around 700 images of 7 denominations of Iraqi banknotes were acquired by a camera. Image augmentation is carried out in order to create a large image dataset that avoids the training model from overfitting and preserves the correct features of the dataset images.

Using different image augmentation techniques, the 700 images were raised to about 7000 images, yielding a dataset for all types of Iraqi banknotes. A rotation with six angles was one of the image augmentation methods. (90,-90,-45, 180, 135,-135,), as well as half crop, and dark (-1.0,-0.5, 0.5) as shown in figure 3.3.

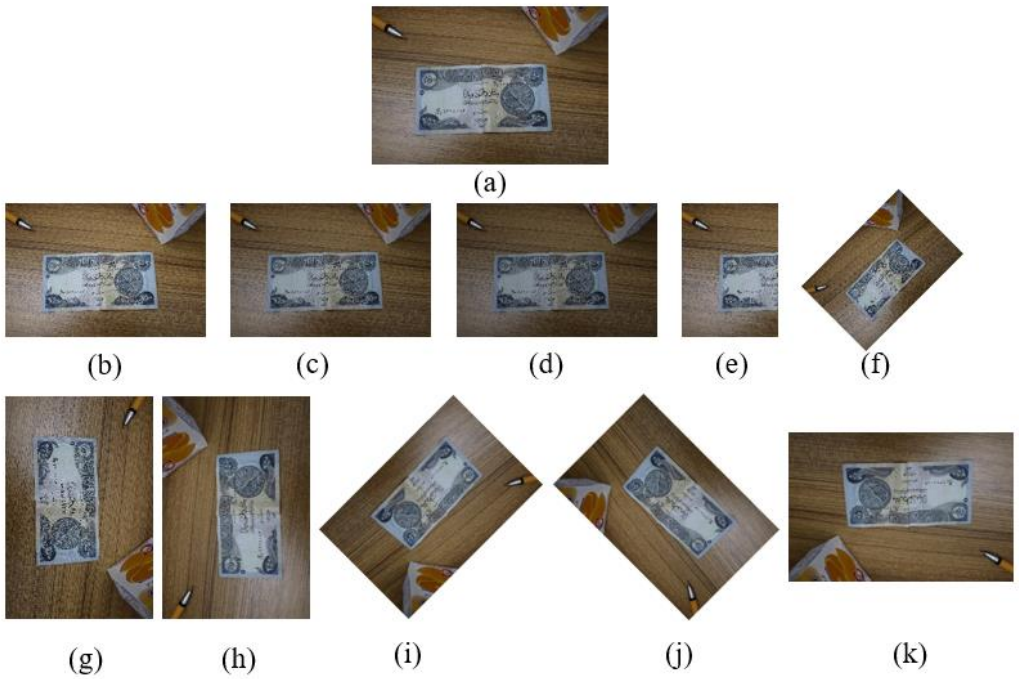


Figure 3.3 Various image augmentation approaches on acquired images (a) Original Image, (b) 0.5 dark, (c) -0.5 dark, (d) -1.0 dark , (e) half crop, (f) -45 angle rotate, (g) 90 angle rotate, (h) -90 angle rotate, (i) 135 angle rotate, (j) -135 angle rotate, (k) 180 angle rotate

Then the "LabelImg" tool was used for labeling the banknotes, and for all dataset images, the accompanying annotation files were saved in the xml format.

Figure 3.4 shows samples of some Iraqi currencies of various denominations that were used for training.

Figure 3.4 Samples of Iraqi currency images in dataset

In addition to these seven items, an additional item labeled as "Iraqi banknote" is constructed of 1000 images and included all the Iraqi currency denominations. This item is included in the main dataset that used to train the main detection model.

The final dataset consists of approximately 44000 images. It is manually examined, and the labeled files are verified to make sure that no errors have been made, and no incorrect data has slipped into the training process. After that, the dataset is split up into two different sub sets, called the training data and the test data. The training data contains around 70% of the data from the entire dataset, and the remaining 30% is used as a test dataset. The distribution of classes and images count is shown in figure3.5.

The OIDv4 toolkit is used to convert XML files of all images in the dataset into three files required for training the YOLOv3 model: train.txt, which contains a list of all training annotation images, one row for each image, test.txt, which contains a list of all test annotation images, one row for each image, and names.txt, which contains a list of all class names.

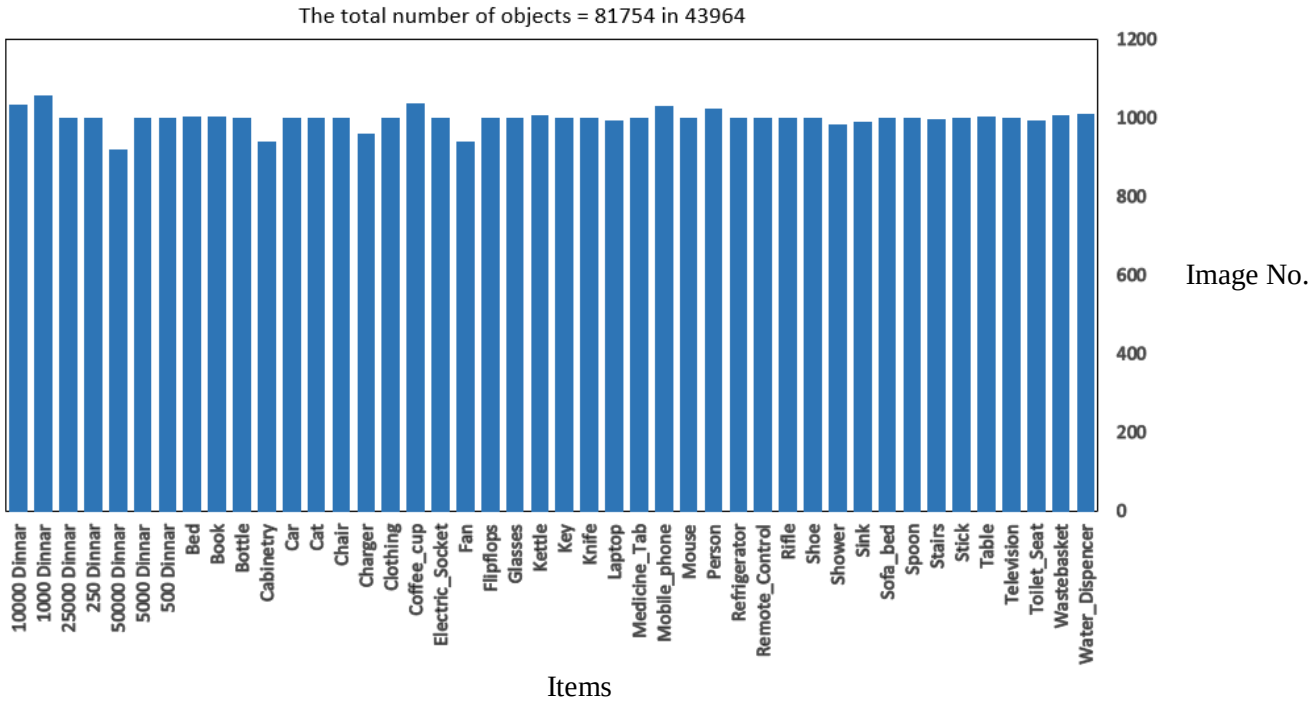


Figure 3.5 Distribution of classes and images count

3.4 Proposed System

Most state-of-the-art methods for detection, such as the YOLOv3 model, work well for what they are designed for, but objects detection faults are discovered during experimentation that yield low accuracy for the system. These faults are discovered when a dataset contains many items and a group of items with approximately the same features relative to the overall items, such as the Iraqi banknotes. To compensate for this, in this thesis, the YOLOv3 multi-stage system has been presented, which is the best alternative to going forward when working on items with similar features.

The proposed design is a YOLOv3 multi-stage system, which is constructed of two stage detections and two YOLOv3 models. Each model is trained on a specific dataset. This design increases the detection accuracy of each stage. The proposed multi-stage system is shown in figure 3.6.

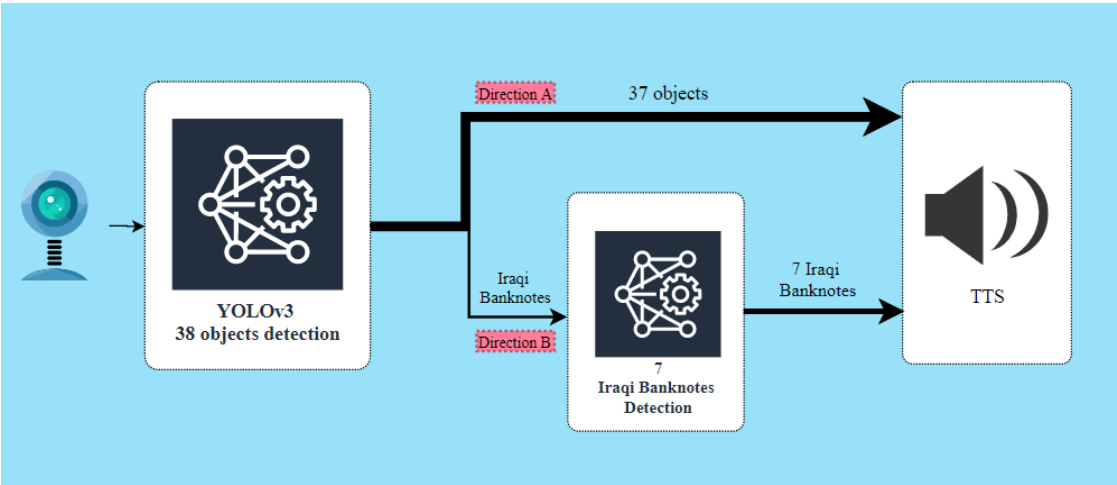


Figure 3.6 Proposed multi-stage system

The first stage of the system is the YOLOv3 objects detection model, which is trained on a dataset containing 38 objects. In the second stage of the system, two directions could be taken: Direction A, direct detection of 37 objects, or Direction B, which is used for Iraqi banknotes detection. When Iraqi banknotes are detected in the first stage, a split occurs and Iraqi banknotes are classified as: "250 Dinar, 500 Dinar, 1000 Dinar, 5000 Dinar, 10000 Dinar", 25000 Dinar, 50000 Dinar".

3.4.1 YOLOv3 Deep Neural Network Implementation

In order to implement an applicable YOLOv3 model and train this model to obtain the proposed objects detection system with state-of-the-art accuracy, and because the official algorithm is deployed on Darknet, which is an open-source neural network framework written in C and CUDA. The Darknet comes with some difficulty with code modification, and it is not the most up-to-date method for exploiting all the capabilities of the GPU. Therefore, the YOLOv3 deep learning neural network is built from scratch on Windows using Python and the TF framework.

To facilitate the process of tracking the work flow of the YOLOv3 deep neural network, the implementation has been divided into several stages, as shown in figure 3.7.

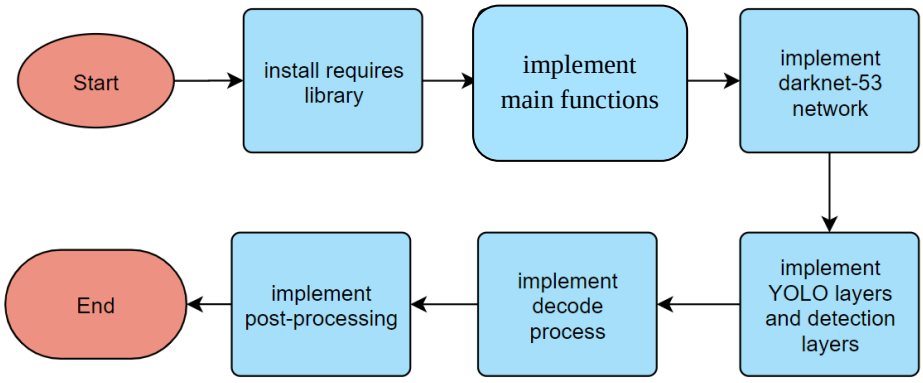


Figure 3.7 Flow diagram of YOLOv3 implementation stages

The implementation begins by writing the code for the YOLOv3 networks, which consists of three different networks. The first network is Darknet-53, which serves as the network's backbone. Secondly, there is an up-sampling network and, lastly, the detection layers, called YOLO-layers. Additionally, the implementation is composed of a decode function for decoding the feature map's and a post-processing function to find the best prediction output. The YOLOv3 network structure is shown in figure 3.8.

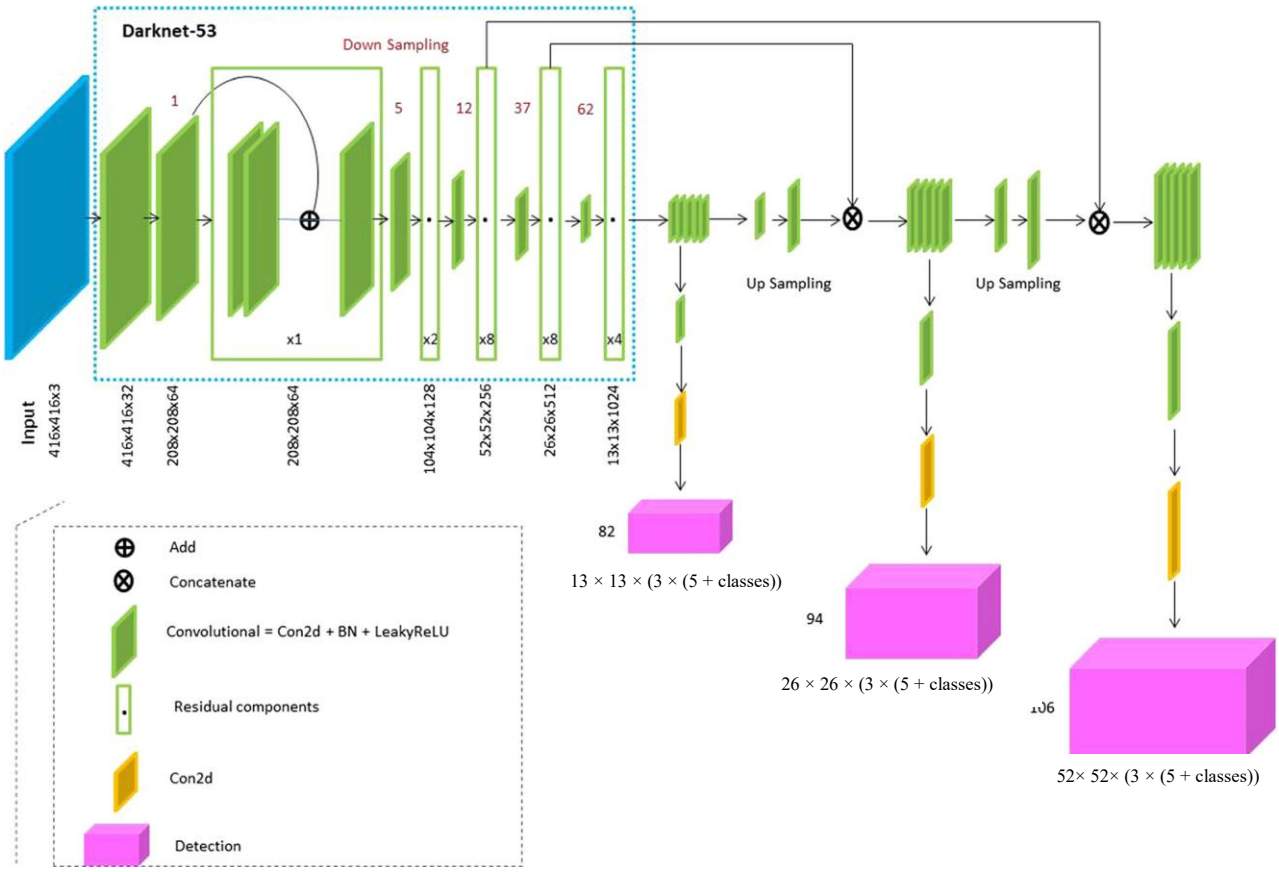


Figure 3. 8 YOLOv3 structure [51]

3.4.2 Main Functions Implementation

YOLOv3 network coding starts by writing the functions of the main layers described in figure 3.8.

Based on the YOLOv3 structure requirements, TF built-in functions such as (convolutional function, a batch normalization function, and the residual function) can't be used directly without modification.

In YOLOv3, there are two types of convolutional layer, with and without a batch normalization layer. The convolutional layer followed by a batch normalization layer uses a leaky ReLU activation layer, as well as in YOLOv3, no form of pooling is used, and a convolutional layer with a stride of 2 is used to down-sample the feature maps. This helps to avoid the loss of low-level features that are frequently attributed to pooling. Therefore, the TF built-in Conv2D function wasn’t used directly and was implemented using a python environment and TF2.3. The flow diagram of the convolutional function is shown in figure 3.9.

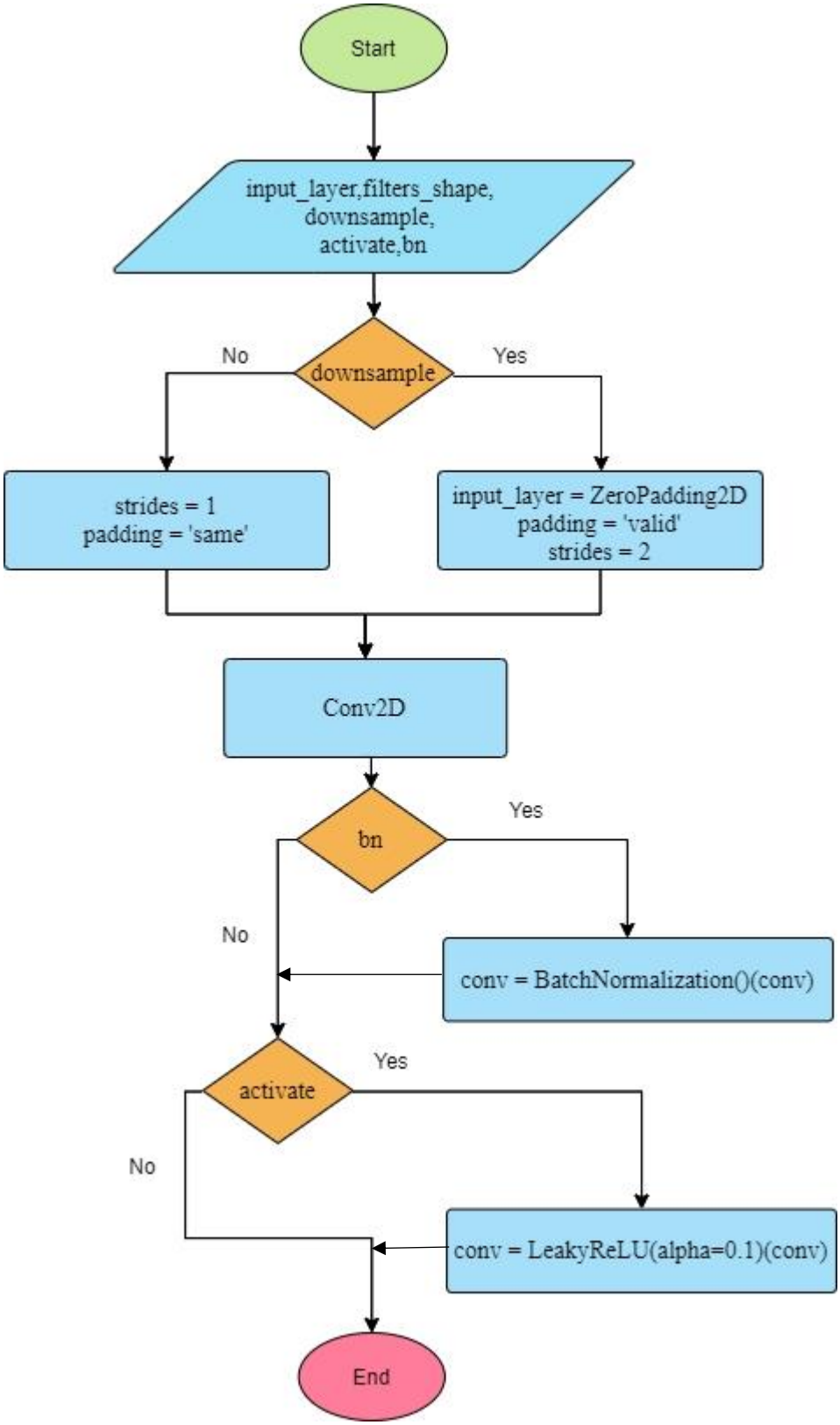


Figure 3.9 Flow diagram of convolutional function

The batch normalization layer is a special case when used with transfer learning during the training of the model. When layer.trainable = False, the batch normalization layer will run in inference mode, and will not update its mean and variance statistics. When the unfreeze model is used, the batch normalization layers should be kept in inference mode by passing variable training = False. Because the updates applied to the non-trainable parameters will destroy what the model has learned. Therefore, the built-in function is modified as follows:

```
Class BatchNormalization(BatchNormalization):
        def call(self, x, training=False):
            if not training:
                training = tf.constant(False)
            training = tf.logical_and(training, self.trainable)
            return super().call(x, training)
```

A residual block is a block consisting of a pair of 3x3 and 1x1 convolutional layers together with a shortcut mechanism to address the gradient disappearance issue caused by expanding the depth of the neural network, the implementation of the residual function is as follows:

```
def  resid(in_lay, in_ch, f_n1, f_n2):
    sh_c = in_lay
    convl = convol(in_lay, flshape=(1, 1, in_ch,f_n1))
    convl = convol(convl, flshape=(3, 3, f_n1, f_n2))
    resid_out = sh_c + convl
    return resid
```

3.4.3 Darknet-53 Network

YOLOv3 is used to detect objects at different scales by constructing a Feature Pyramid Network (FPN). Darknet-53 is an efficient backbone for performing feature extraction with a very deep network that contains 53 layers, which are mainly composed of convolutional and residual blocks.

Every convolutional layer in Darknet-53 is followed by the Leaky ReLU activation function and batch normalization layer. A full overview of the Darknet-53 architecture is shown in table 3.2. The previously coded functions are used in building the Darknet-53 (for more details, see Appendix-C for the Darknet-53 coding).

Table 3.2 Darknet-53 architecture

Type	Filters	Size/Stride	Output	
Convolutional	32	3 × 3/1	416 × 416 × 32	
Convolutional	64	3 × 3/2	208 × 208 × 64	
Convolutional	32	1 × 1/1	208 × 208 × 32	
Convolutional	64	3 × 3/ 1	208 × 208 × 64	×1
Residual			208 × 208 × 64	
Convolutional	128	3 × 3/2	104 × 104 × 128	
Convolutional	64	1 × 1/1	104 × 104 × 64	
Convolutional	128	3 × 3/1	104 × 104 × 128	×2
Residual			104 × 104 × 128	
Convolutional	256	3 × 3/2	52 × 52 × 256	
Convolutional	128	1 × 1/1	52 × 52 × 128	
Convolutional	256	3 × 3/1	52 × 52 × 256	×8
Residual			52 × 52 × 256	
Convolutional	512	3 × 3/2	26 × 26 × 512	
Convolutional	256	1 × 1/1	26 × 26 × 256	
Convolutional	512	3 × 3/1	26 × 26 × 512	×8
Residual			26 × 26 × 512	
Convolutional	1024	3 × 3/2	13 × 13 × 1024	
Convolutional	512	1 × 1/1	13 × 13 × 512	
Convolutional	1024	3 × 3/1	13 × 13 × 1024	×4
Residual				

3.4.4 Up-Sampling Layers and YOLO Layers

The up-sampling layers and the three YOLO layers used the features extracted by Darknet-53 to detect objects at three different scales. These scales are 1/32, 1/16 and 1/8 of the input size. Figure 3.10 shows the architecture of up-sampling layers and YOLO layers.

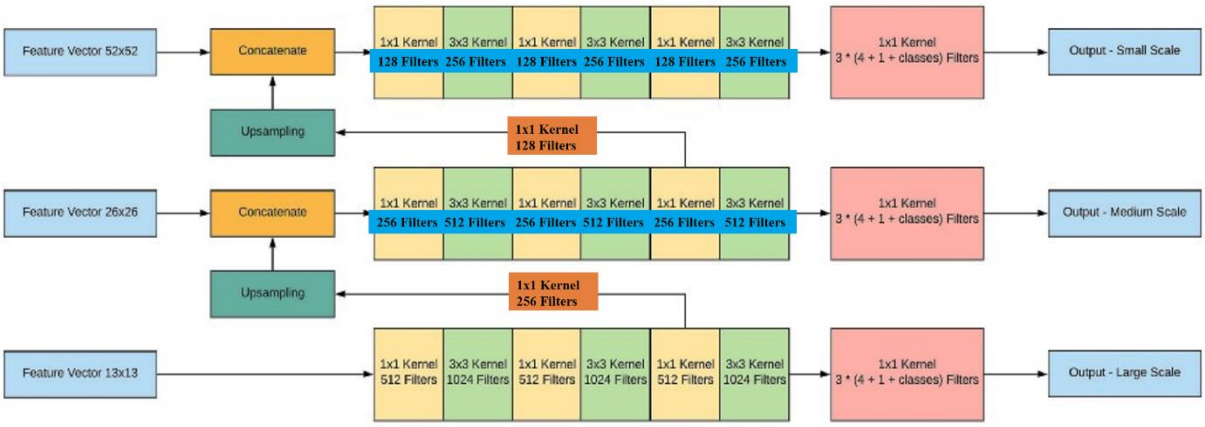


Figure 3.10 Up-Sampling layers and YOLO layers

The up-sampling method is used in order to concatenate with the shortcut outputs from Darknet-53 before applying detection on a different scale. The feature map is up-sampled using nearest neighbor interpolation. The up-sample function was coded as follows:

```
def upsample(in_layer):
    return tf.image.resize (in_layer, (in_layer.shape [1] * 2, in_layer.shape [2] * 2), method='nearest')
```

Each YOLO layer consists of a number of convolutional layers. The YOLO layer is coded using a convolution function that is coded in section 3.3.2 (for more details, see Appendix-C for the YOLO layers coding), and the parameters of the convolution layers are set as mentioned in figure 3.10.

The parameters of the detection layer are designed to vary depending on the number of classes used in a model. The detection is performed on three different scales. The first scale produces a tensor of size $13 \times 13 \times (3 \times (5 + classes))$, which was produced by down sampling the input into 13×13 and produce a detection on layer 82^{th}. Then, in order to perform the second detection scale, produce a tensor of size $26 \times 26 \times (3 \times (5 + classes))$. The feature map is created from layer 79 by applying it to a convolutional layer before being up-sampled by a factor of two to have a size of 26×26. Then concatenate the up-sampled feature map with the layer 61 feature map. After that, the concatenated feature map is submitted to a few additional convolutional layers till the second detection scale is done on layer 94. Layer 106 is used for the final detection layer, resulting in a tensor of size. $52 \times 52 \times (3 \times (5 + classes))$. The feature map from layer 91 is applied to a convolutional layer and then concatenated with a feature map from layer 36. After that, the combined feature map is submitted to a few additional convolutional layers till the third detection scale is obtained.

Each scale have $(3 \times (5 + Classes))$ entries in the feature map. Three represents the number of bounding boxes each cell can predict. According to the paper [51], each of these three bounding boxes specializes in detecting a certain kind of object. Each of the bounding boxes have $(5 + Classes)$ attributes, which describe the center coordinates, the dimensions, the object score and class confidence for each bounding box.

3.4.5 Decoder Function

YOLO does not predict the absolute coordinates of the bounding box, object score and class confidence. Therefore, the output of the YOLOv3 network is given to the decode function, which decodes the channel information on the feature map. The following formulas describe how the decode function transforms the network output to obtain bounding box predictions (for more details, see Appendix-C for the decode function coding)[51]:

$$b_x = \sigma(t_x) + c_x \quad \text{.....} \quad (3.1)$$

$$b_y = \sigma(t_y) + c_y \quad \text{.....} \quad (3.2)$$

$$b_w = p_w e^{t_w} \quad \text{.....} \quad (3.3)$$

$$b_h = p_h e^{t_h} \quad \text{.....} \quad (3.4)$$

Where:

t_x, t_y, t_w, t_h : Output of the network.

b_x, b_y: x, y Center coordinates of prediction box.

b_w, b_h : Width and height of prediction box.

c_x , c_y : Top-left coordinates of the grid.

p_w , p_h : Dimensions of the anchor box.

The decode function calculates the center coordinates using a sigmoid function. This procedure forced the output value to be between zero and one, and the result was then added to the grid cell's top left corner coordinate that predicted the object. Then the bounding box dimensions are predicted by applying a log-space transform to the result and multiplying it by an anchor. In YOLOv3, the anchor box is regarded as the a priori bounding box, and constraints are imposed on the predicted bounding box.

Additionally, the decode function is used to calculate the object score and class confidence by running the predicted output through a sigmoid and interpreting the result as a probability.

3.4.6 YOLOv3 Post Processing

YOLOv3 predicted $((52\,x\,52) + (26\,x\,26) + (13\,x\,13))\,x\,3 = 10647$ bounding boxes in a 416×416 image. To address the issue of multiple detections of the same object, these detection bounding boxes have been reduced.

As shown in figure 3.11, the post-processing function is used to reduce bounding boxes, which takes the predicted list of bounding boxes, a threshold score parameter, and the original image size.

The bounding box attributes have been $, b_y, b_w, b_h$, but because it was easier to work on the coordinates of two points: the top left and the bottom right. Therefore, the output has been converted to this format. Then the boxes are scaled to fit into the original image shape.

The reduction procedure begins with a threshold of object confidence, which is used to remove boxes with low confidence scores. Then to avoid having multiple boxes for the same object, a second filter named NMS was used to select the appropriate boxes for each class.

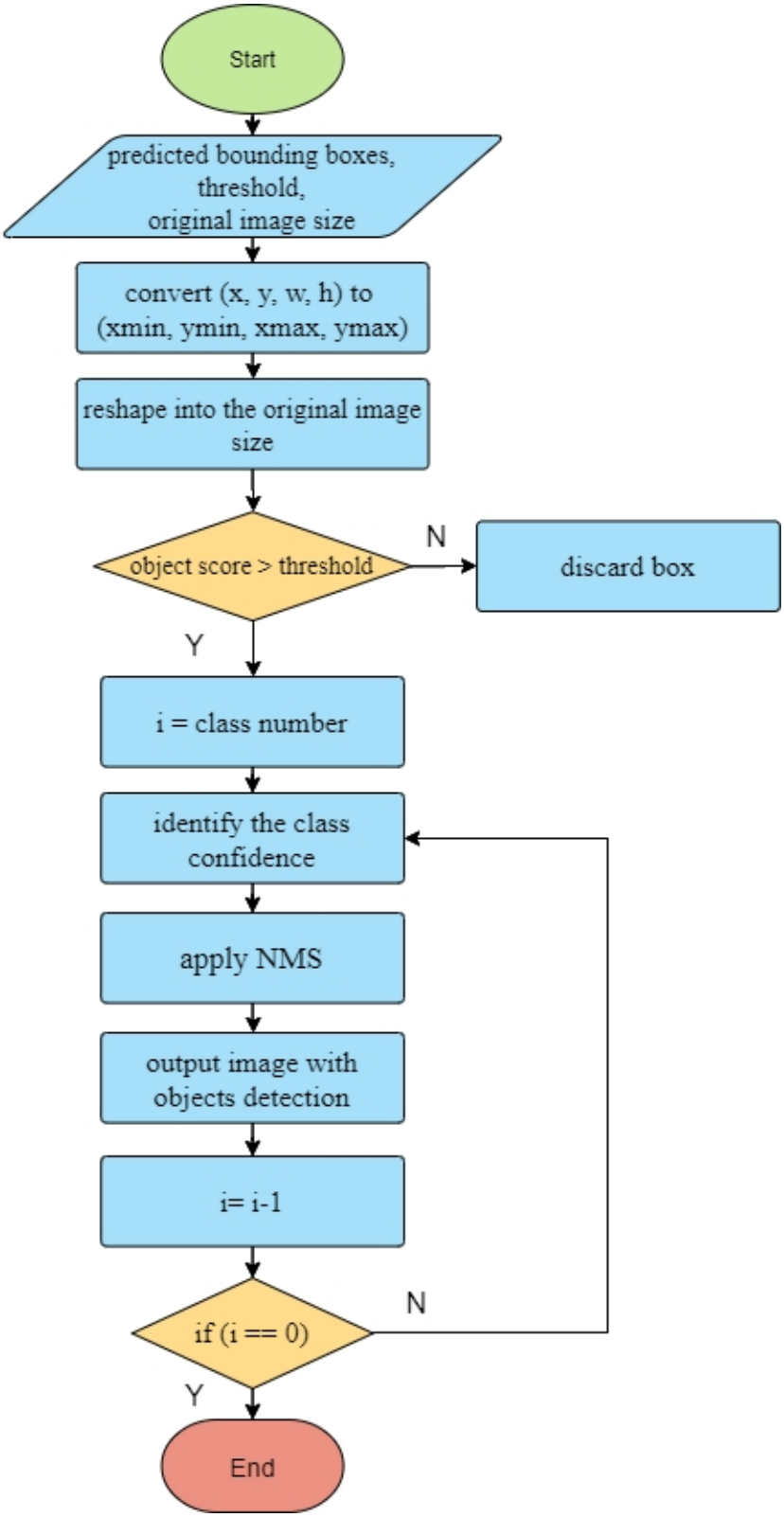


Figure 3.11 Procedure of YOLOv3 post-processing

The following are the key steps for implementing the NMS iterative process:

- Stop iteration, if number of bounding boxes equal zero.
- Sort the bounding box by score order.
- Compute the IoU of the first bounding box in respect to all of the other bounding boxes, and remove any bounding boxes with IoU values that exceed the threshold.
- Tag the first bounding box as a selected box, repeat the above steps.

The IoU is utilized by NMS to eliminate all the boxes that had a substantial overlap with the boxes that had previously been selected. Only the best boxes remain.

The following steps are the key to IoU implementation:

- Calculated the area of the two boxes.

  $Box_1Area = B_1y_2\text{-}B_1y_1 * B_1x_2 \text{-}B_1x_1$

  $Box_2Area = B_2y_2\text{-}B_2y_1 * B_2x_2 \text{-}B_2x_1$
- Found the coordinates of the intersection of the two boxes:
  - xi_1 = max of the two boxes' x1 coordinates.
  - yi_1 = max of the two boxes' y1 coordinates.
  - xi_2 = min of the two boxes' x2 coordinates.
  - yi_2 = min of the two boxes' y2 coordinates.
- Calculate intersection area

  $IntersectionArea = yi_2 – yi_1 * xi_2 – xi_1$

- Calculate IoU

  $IoU = IntersectionArea / (Box_1Area + Box_2Area - IntersectionArea)$

3.5 Dataset Preprocessing

After formatting annotations to match the requirements of the YOLOv3 model and before any training, it is important to preprocess the images and the annotations of each image in the dataset, as shown in figure3.12. Image preprocessing starts by verifying the annotation file to ensure that the image path is valid and that the image already exists, to ensure that no incorrect data enters the training process. To avoid overfitting, image augmentation has been used to increase the variety of the dataset. Then the image is resized to the pre-specified shape by keeping the aspect ratio, and padding the left-out portions, and updating the ground-truth box.

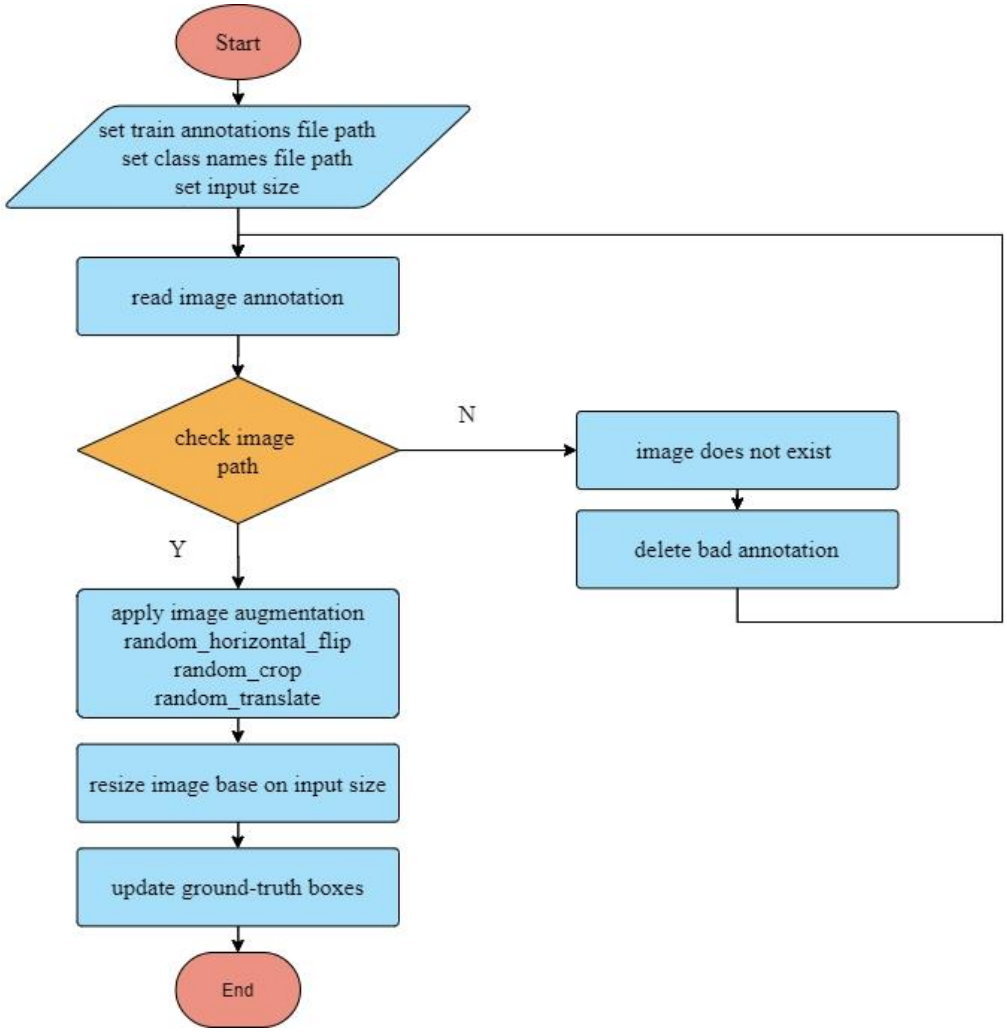


Figure 3.12 Dataset preprocessing flow diagram

3.6 Optimizer Type and Learning Rate (LR)

The LR and optimizer types are hyper parameters with the greatest impact on performance. When working on deep learning, choosing the hyper parameters is a common issue. In training, the optimizer type and learning rate are chosen manually. It is time-consuming, but it makes the overall system achieve better results in the long range.

3.6.1 Optimizer Type

According to most studies like [65], stochastic gradient descent (SGD) causes divergence very quickly, but the adaptive moment estimation optimizer (ADAM) always seems to converge. Therefore, the ADAM optimizer is used as the default optimizer in training.

One of the key differences between the Darknet implementation and the current implementation is that the Darknet used a (SGD) optimizer.

3.6.2 Learning Type

There are two types of learning; traditional machine learning techniques which try to learn each task from scratch, and transfer learning techniques which try to transfer the knowledge from some previous tasks to a target task when the latter has less training data. Even though the model was trained on a custom dataset, it was advantageous to use another already trained model's weights as a starting point.

Weights of the network are initialized with pre-trained using transfer learning in the coco dataset and according to the study [66] [67], the cosine decay learning rate had better performance with transfer learning. Therefore, it is used instead of the step learning rate schedule which is used in Darknet YOLOv3.

3.6.3 Learning Rate

The challenge of training deep learning neural networks involves carefully selecting the learning rate. The learning rate is an adjustable hyper parameter that usually has a positive value, often greater than 0.0 less than 1.0. but in deep learning studies the range of learning rate always between 1E-1 to 1E-6, because the deep neural network deal with a large amount of input, and it is very unstable at the start of training, to guarantee that the network may have great convergence, the weights should change less aggressively and the learning rate should be set extremely low at first.

To choose a proper initial learning rate and final learning rate for the proposed system experiments done using 20 % of training data to increase the speed of this process for 5 epoch each time. Experiment results in table 3.3 show the inability of the model to learn anything at the too-large learning rates (1E-1, 1E-2 and 1E-3) and the model is able to learn well with the learning rates (1E-4, 1E-5 and 1E-6) although successively slower as the learning rate was decreased. The results suggest a moderate learning rate of 1E-4 results in good model performance on the training set.

Table 3.3 Best initial learning rate

Total Loss						
LR	Start	Epoch 0	epoch 1	epoch 2	epoch 3	epoch 4
1.00E-01	2299.71	nan	nan	nan	nan	nan
1.00E-02	1989.59	nan	nan	nan	nan	nan
1.00E-03	2483.27	nan	nan	nan	nan	nan
1.00E-04	1976.87	30.55	26.72	25.51	24.25	23.11
1.00E-05	2136.22	107.96	41.86	38.89	36.91	33.42
1.00E-06	2064.12	894.8	274.37	95.42	55.46	43.5

3.6.4 Warm Up and Scheduler Cosine Decay

Learning rate warm up is applied to progressively increase learning rate from 0 to the initial learning rate linearly and reach network training's "warm-up" stage. The learning rate has been changed to the following formula:

$$LR = \frac{i \times \eta}{m} \quad (1 \leq i \leq m) \quad \text{.....} \quad [65](3.5)$$

Where

m: first batches (data for 5 epoch) to warm up.

η : Initial learning rate.

i: batch number.

However, if the loss of network training is reduced, it is not appropriate to always utilize a higher learning rate, since this would cause the gradient of the weight to fluctuate, making it impossible to achieve the global minimum loss of training. Therefore, **scheduling as cosine decay** reduction method has been chosen, which has proposed a simplified version for decreasing the learning rate from the initial value to end learning rate by following cosine function where the learning rate η_t is computed as:

$$\eta_t = \propto + \frac{1}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)(\eta - \propto) \quad \text{.....} \quad [65](3.6)$$

Where

T: total number of epochs. (The warmup stage is ignored).

t: current epoch. (The warmup stage is ignored)

η : initial learning rate

$\propto$: end learning rate.

Figure 3.13 shows Visualization of learning rate schedules with warm-up. Then cosine decay decreases the learning rate slowly at the beginning, and then becomes almost linear decreasing in the middle, and slows down again at the end.

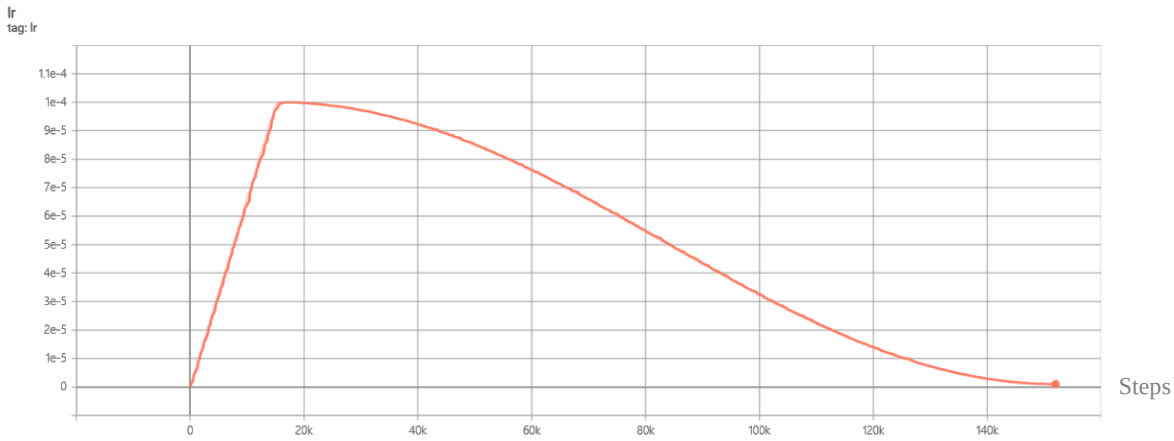


Figure 3.13 Visualization of learning rate scheduler cosine decay with warm-up

3.7 YOLOv3 Loss Function

The loss function is used to measure a model's performance, in terms of being able to predict the expected outcome. The loss function of YOLOv3 consists of three different parts, which are:

Confidence loss: Check to see if the prediction frame contains any objects. The confidence loss for detection or non-detection in a block is:

$$L_{conf} = \sum_{i=0}^{s^2} \sum_{j=0}^{B} \zeta_{ij}^{obj} (c_i - \breve{c}_i)^2 + \gamma_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \zeta_{ij}^{noobj} (c_i - \breve{c}_i)^2 \ldots [68](3.8)$$

Where:

ζ_{ij}^{obj} = 1: if there is object in the block

c_i : Ground truth box confidence in the block i.

$\breve{c}_i$: Prediction box confidence in the block i.

ζ_{ij}^{noobj} =1: if there is no object in the block.

B: number of anchor boxes.

S: grid cell.

The rule for calculating loss of confidence is as follows: it is considered to be a background if the IoU of a prediction box and all real boxes is less than a specified threshold. It's the foreground else (including objects).

Classification loss: Identifying the object category in the prediction frame. The classification loss in each block is the squared error for each class:

$$L_{clas} = \sum_{i=0}^{s^2} \zeta_i^{obj} \sum_{c \in classes} (p_i(c) - \breve{p}_i(c))^2 \quad .... \quad [68] (3.9)$$

Where:

$\breve{p}_i(c)$: denotes the conditional class probability for class c in block i.

$p_i(c)$ =1 if the ground truth is c, otherwise is zero.

The total classification loss is the sum-up of losses in each grid S. The cross-entropy of the two classes is utilized as the classification loss. That is, all category's classification problem is reduced to whether it belongs to this category. As a result, multi-classification is considered a two-classification issue. The benefit of this is that it eliminates the mutual exclusion of the categories, mainly to address the issue of missing detection due to the overlapping of multiple categories of objects.

Box regression loss: When there are objects in the prediction box, the value is computed. The localization error is a measurement of how well the prediction and ground truth bounding boxes overlap:

$$L_{regr} = \gamma_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \breve{x}_i)^2 + (y_i - \breve{y}_i)^2 \right] +$$
$$\gamma_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\breve{w}_i})^2 + (\sqrt{h_i} - \sqrt{\breve{h}_i})^2 \right] \quad \ldots\ldots\ldots \quad [68](3.10)$$

Where: γ_{coord} is the regularization term for the bounding box coordinates. There can be multiple bound box B in one grid S. The bounding box coordinate is predicted by knowing its box center $(\breve{x}, \breve{y})$ and its height and width $(\breve{w}, \breve{h})$, where x, y, h, w is the ground truth. The regression loss was coded as follows:

```
respbox  = label[:, :, :, :, 4:5]
bboxesloss = 2.0 - 1.0 * labelxywh[:, :, :, :, 2:3] * labelxywh[:, :, :, :, 3:4] /
(in_size ** 2)
gioulos = respbox * bboxesloss * (1 - giou)
gioulos = tf.reduce_mean(tf.reduce_sum(gioulos, axis=[1,2,3,4]))
```

Where respbox denotes that if the grid cell contains objects, the bounding box loss will then be calculated. A high GIoU value between the two bounding boxes indicates a low loss value. The network will improve in the direction of more overlap between the prediction box and the real box. In this thesis implementation, the original IoU loss was substituted with the GIoU loss. The benefit of GIoU is that it enhances the distance measurement method between the anchor box and the prediction box.

The final object detection loss for YOLOv3 is the sum of the confidence, classification, and regression losses.

3.8 Converting Pre-trained Weights to TF Format

To copy all the pre-trained weights from the original Darknet layers, which was trained on the COCO dataset, to the YOLOv3 network layers, the TF build in function has not been used to load pre-trained model weights because the layer ordering on tf.keras and the Darknet is different. Therefore, the load weights function is coded from scratch. The flow diagram of it is shown in figure 3.14.

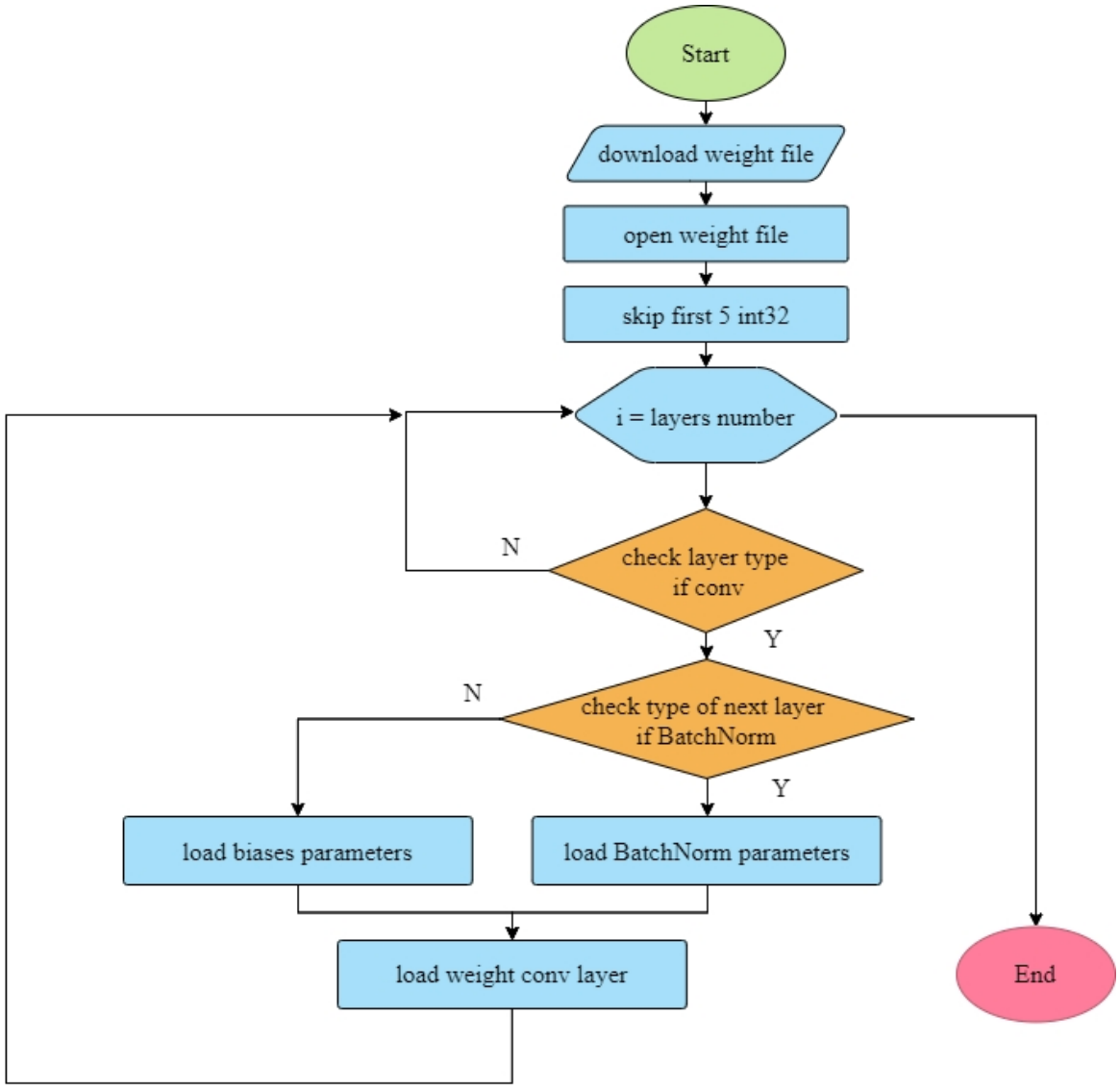


Figure 3.14 Load weights function flow diagram

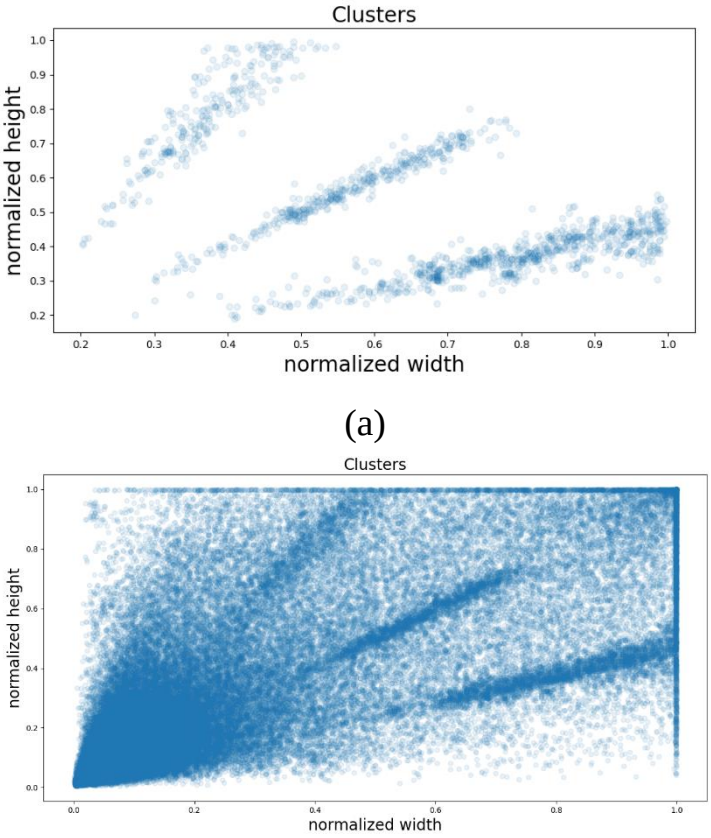
The process starts by opening the file, and skipping the first 5 int32 values, which are the header information (major version number, minor version number, subversion number) followed by int64 value which is number of images that the network trained by. After that, start from the first convolutional layer, then check the type of the layer following the one currently processed and read the appropriate number of values. If it is followed by a batch normalization layer, load weights of the batch norm layer, then weights of conv layer. In the opposite case, when the conv layer is not followed by the batch norm layer, instead of reading batch norm parameters, read bias weights. And repeat these steps to the last layer.

3.9 Custom Anchor Boxes

In order to pre-specify the shapes of anchor boxes, YOLOv3 proposes using the K-means clustering algorithm on bounding box shapes to find the proper anchor boxes for the data. K-means clustering is used to discover suitable candidate anchor boxes. Using the direct Euler distance metric, K-means minimizes errors for bigger bounding boxes but not for smaller boxes. Therefore, YOLOv3 used IoU as a distance metric.

To prepare the data features for K-means clustering, standardized the bounding box width and height with the image width and height, the coordinates of a bounding box, x and y, are not of concern. IoU needs to only compare the shapes of the bounding boxes. The IoU calculations are made assuming all the bounding boxes are located at one point, and only width and height are used as features.

Figure 3.15 shows the height and width plotted against each other for Iraqi banknotes dataset and 38 objects dataset.



(a)

(b)

Figure 3.15 Height and width plotted against each other for
(a) Iraqi banknotes dataset (b) 38 objects dataset

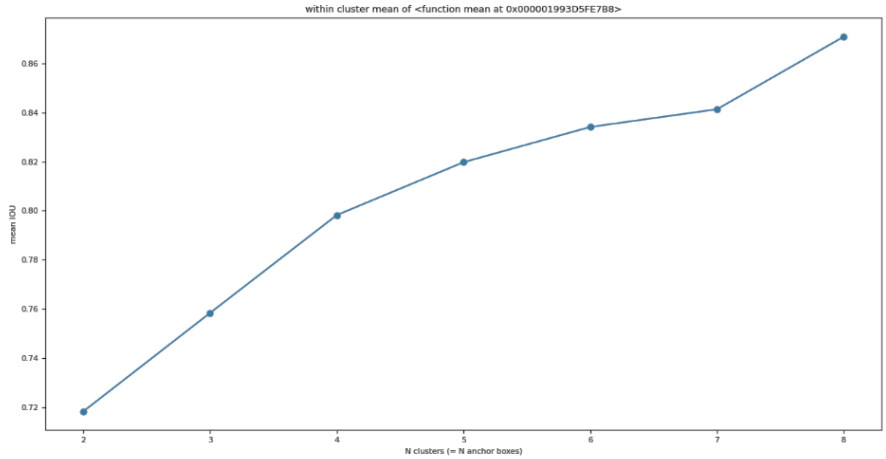
K-means clustering is used by specifying the number of clusters and establishing the cluster centers which consists of two basic steps:

Step 1: Assigning each item to the cluster center that is closest to it. The distance to the cluster center is computed using a 1-IoU scale.

Step 2: Determine the cluster centers by taking the median of all cases in the clusters. Steps 1 and 2 should be repeated until the two competing iterations produce identical cluster centers.

The mean of the maximum IoU between the bounding box and individual anchors is computed using 9 clusters to stay true to the original YOLOv3 implementation. Figure 3.16 presents IoU vs. number of cluster centers data.

(a)



(b)

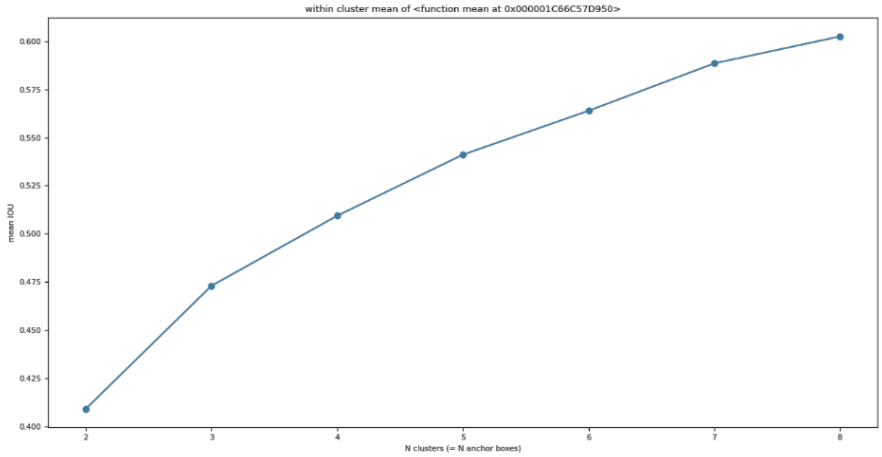


Figure 3.16 Mean IoU vs cluster centers data for (a) Iraqi banknotes, (b) 38 objects.

After plotting the number of cluster centers vs mean IoU, it is clear that the more clusters, the larger mean IoU becomes and at 9 anchor boxes, the mean IoU was above 87%. Since the results of the K-means algorithm are sensitive to the selection of the initial points, the results of each run are not the same.

The k-means algorithm to generate nine anchor boxes based on the 38 objects dataset through the calculation these nine clusters are: (21, 33), (76, 76), (59, 260), (199, 107), (160, 334), (123, 188), (381, 371), (270, 334), (302,201). And the k-means algorithm to generate nine anchor boxes based on the Iraqi banknotes dataset through the calculation these nine clusters are :(218, 105), (142, 168), (283, 131), (139, 293), (215, 216), (331, 151), (176, 372), (390, 180), (273, 272).

3.10 Training and Evaluating the YOLOv3 Model

Deep learning training takes a long time due to the large amount of data (images and annotations) in the dataset and the complexity of the YOLOv3 deep network structure. The GPU is utilized to train instead of the CPU to solve this problem. Also, an optimization for the change in learning rate is implemented during training.

The model is trained on the labeled images via continuous iteration for 100 epochs using forward and backward propagation with the Adam optimizer and the model configuration. The weights are stored during the training process after each epoch. After training, the model is utilized to generate detection in real-time live video, making detection requires the use of the YOLOv3 model and the specific weights trained with that model. The text class prediction of the objects detected in each frame is converted to voice feedback using the Google TTS API. Figure 3.17 depicts the training and detection workflows.

To assess the performance of the YOLOv3 model, the test dataset is used to compute the AP for each class and average those to calculate the mAP (mentioned in 2.9). Detections are considered TP if the IoU between the predicted boundary and the real object boundary exceeded the threshold value, and are deemed FP if the IoU did not exceed the threshold value.

Finally, when there is a ground truth in the image and the model failed to detect the object, the detection is considered an FN.

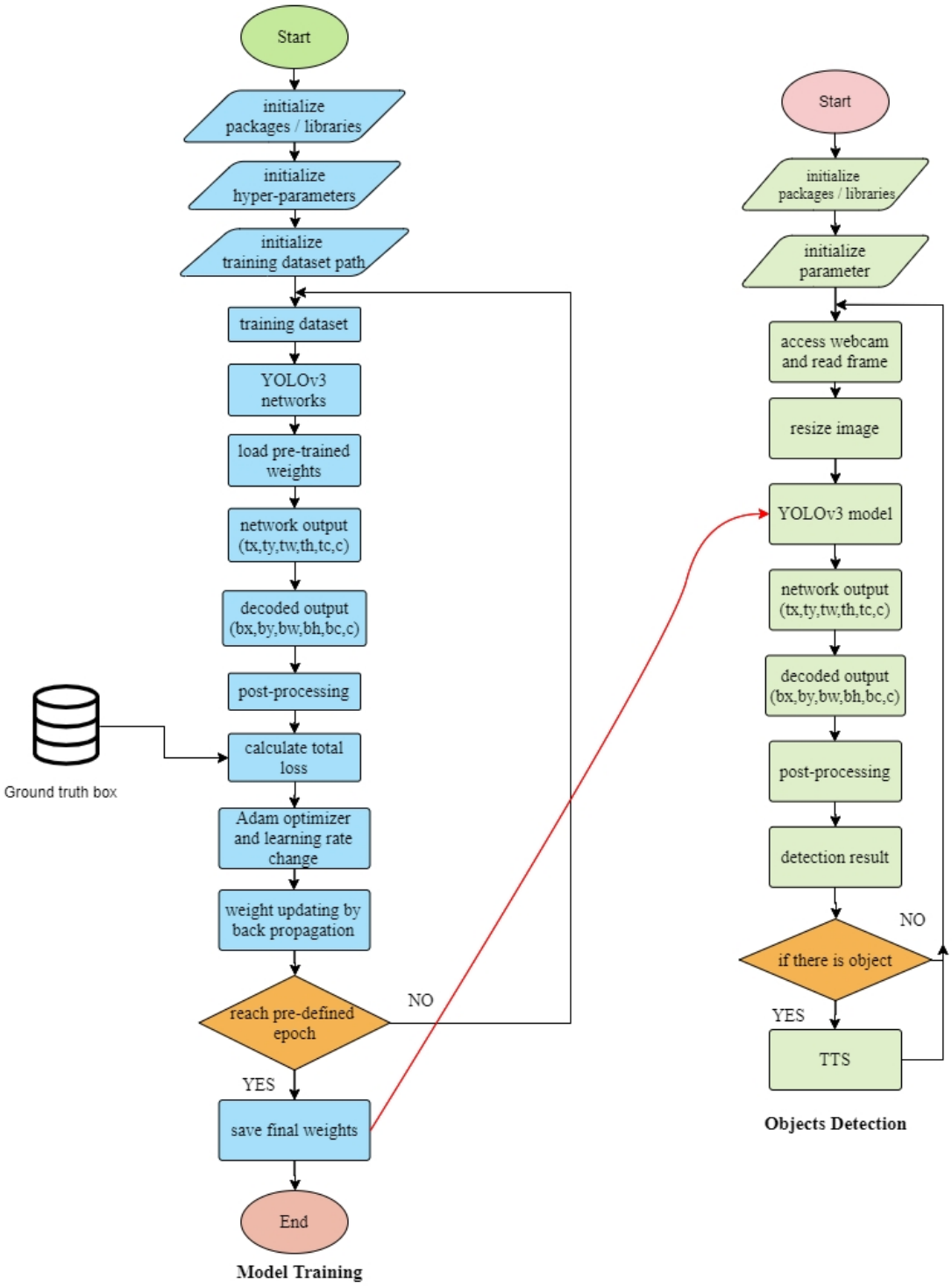


Figure 3.17 Training and detection workflows

CHAPTER FOUR

Analysis of System Results

4.1 Introduction

In this chapter, the results obtained after a series of experiments conducted in order to develop the proposed system and achieve the desired accuracy and detection time will be presented. A synthesis of theory and experimentation has been used. This combination of theory and experimentation led to well-structured experiments with deep theoretical foundations.

Alot of time is needed to exhaustively create and test the model. The implementation of the proposed system includes the following stages: dataset construction and labeling stage, implementation of the YOLOv3 deep neural network using Python and TF stage, model training stage, evaluation stage, improvement stage, and inference stage. The final system has been tested in real-world conditions using live video, which was an important step towards successful implementation.

The proposed system is designed in such a way to be general that it can be extended to detect any objects.

4.2 Training Results

The YOLOv3 model has been trained from scratch using a dataset consisting of 44 object classes and approximately 44000 images (which are mentioned in table 3.1) and hyper parameters shown in table 4.1.

The batch size was set to 1 and the GPU used during training was an NVIDIA GeForce GTX 1650 (as mentioned in 3.1). Any higher batch size would lead to a CUDA out of memory error.

Table 4.1 Hyper parameters

Name	Value
Epoch number	100
Batch size	1
Train LR initial	1.00E-04
Train LR end	1.00E-06
Warm-up epoch	2
Input size	416
Learning policy	Adam

In this thesis, the optimization is done successfully for overall training time by changing the value of learning rate adaptively with the train loss value. The change in learning rate is based on cosine decay and warm-up. In the course of the training, the loss was saved in the log, and TensorBoard was used to clearly visualize the progress of the entire training.

Training was consumed for a very long time, approximately a month, then stopped after 100 epochs. Figure 4.1 depicts the TensorBoard training loss progress.

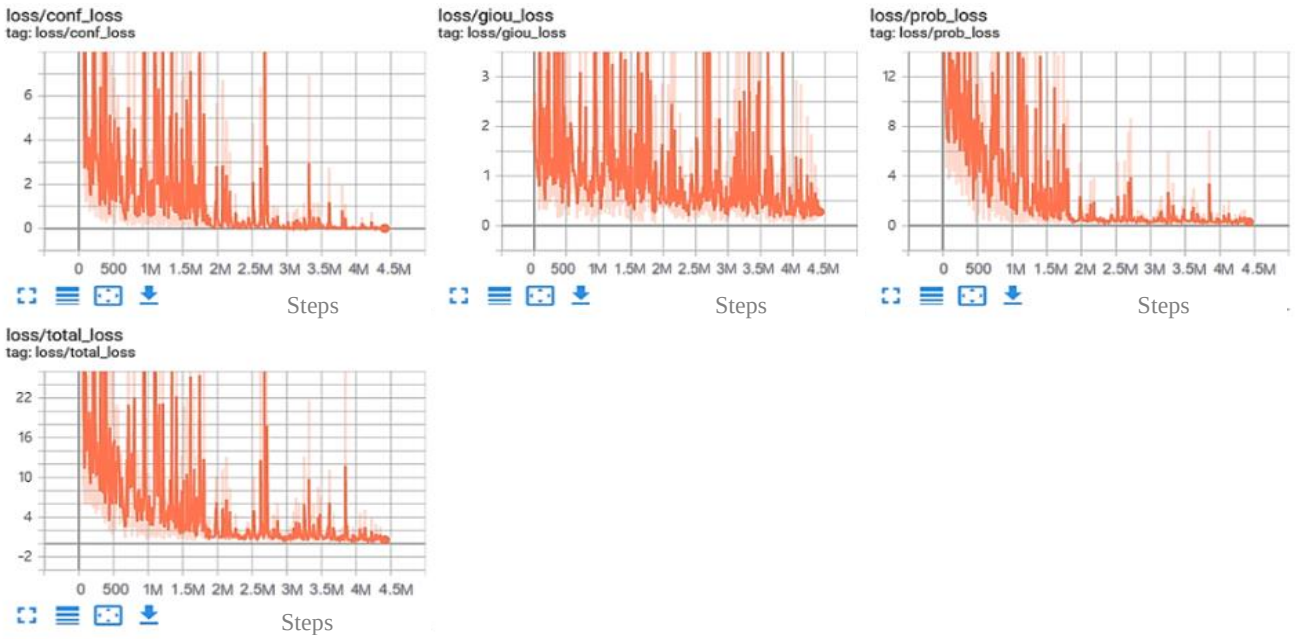


Figure 4.1 The progression of the training loss is depicted by TensorBoard

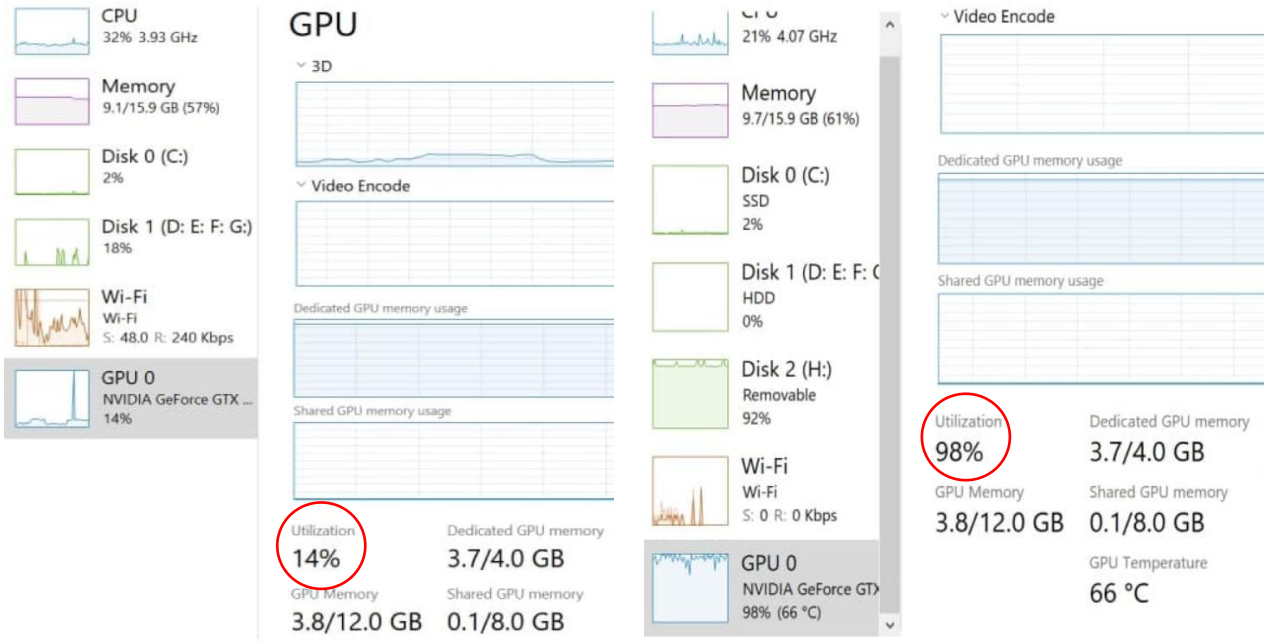
The model performance was evaluated by computing the mAP metric with a 0.5 threshold value for the test dataset. The mAP result value was 8.564%, with an average evaluation time of 12.17 FPS.

4.3 Optimization

Various optimization methods have been tried and their effect on performance has been evaluated, including process time optimization, size of the input layer, transfer learning, YOLOv3 multi-stage detection and creating custom anchors.

4.3.1 Process Time Optimization

The GPU performance priority on the window was set to high and utilized all GPU capabilities. This reduces the time spent on training the YOLOv3 model to about two weeks. Figure 4.2 (a, b) depicts the impact of changing the priority on GPU performance.



(a) (b)

Figure 4.2 GPU performace (a) before, (b) after

4.3.2 Size of the Input Layer

To illustrate the impact of changing the size of the input layer of the YOLOv3 network on the accuracy and speed of a model, further experiments were carried out. Table 4.2 illustrates the relationship between speed, mAP and the input layer size.

Table 4.2 Illustrating the relationship between speed, mAP and the input layer size

Input Layer Size	mAP	FPS
320 × 320	4.267%	15.65
416 × 416	8.564%	12.17
512 × 512	10.654%	10.85

From the results in Table 4.2, one can observe a proportional increase in detector accuracy and a decrease in speed in relation to the increase in the input layer size.

An explanation for this is that the input layer is mapped to the resolution of the image to be processed. Larger images contain more data and details for the model to process, but require more computing power. However, these details lack important key points that have a significant impact on the model's detection accuracy.

In a scenario where speed is not important, the input layer could be increased to have a small impact on accuracy. However, in such cases, it is better not to use YOLO at all but an object detector which is designed for slow but accurate detection, such as the Fast R-CNN.

In order to balance between speed and accuracy, the best input size chosen for the proposed system was 416 × 416.

4.3.3 Transfer Learning

The YOLOv3 network training process with the complete dataset, which consists of about 44000 images, is time-consuming. Therefore, in order to minimize the time during the experiments, 20% of the dataset images were used. This reduces the time spent on training the YOLOv3 model to about 4 days.

Transfer learning from a pre-trained model has been utilized and evaluated. The COCO pre-trained model was chosen to transfer knowledge from it, because it was trained on a Google image dataset, and the relied on dataset consisted of 27% of the images were taken directly from the Google library, while the rest of the images were close to the images in the Google library.

The examination results found that transfer learning yields the highest increase in the accuracy of the YOLOv3 objects detector, which achieves a mAP of 73.403% and 12.53 FPS.

This means that the time required in the deep learning process to produce an accurate result can be reduced by the careful use of transfer learning. Figure 4.3 shows objects' AP before and after using transfer learning.

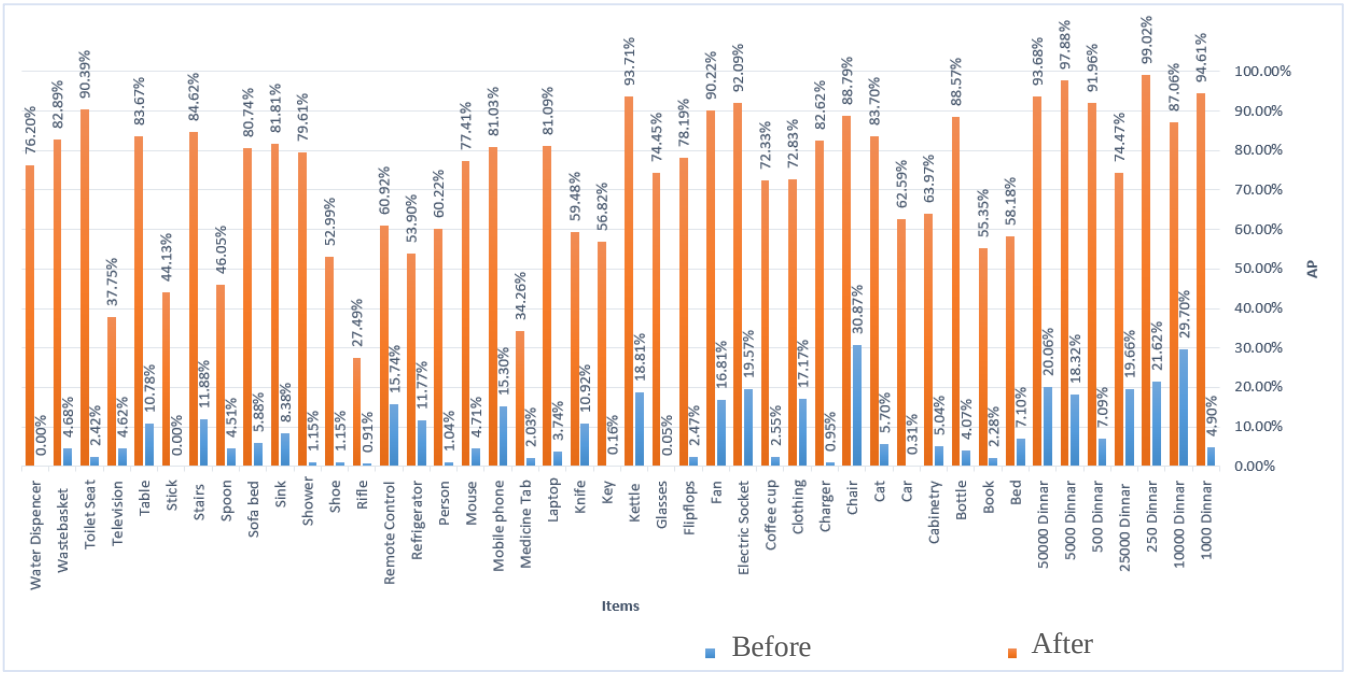


Figure 4.3 Objects' AP before and after transfer learning

4.3.4 YOLOv3 Multi-Stage Detection

Through careful observation of the AP of items resulting from previous experience, (Bed, Book, Key, Knife, Medicinal Tab, Refrigerator, Rifle, Shoe, Spoon, Stick, Television) have an accuracy less than 60%, while (1000 Dinar, 250 Dinar, 500 Dinar, 5000 Dinar, 50000 Dinar, Electric Socket, Fan, Kettle, Toilet Seat) have an accuracy more than 90%.

Noted that most of the items that got high accuracy were Iraqi banknotes, which consist of seven items and have approximately the same features. That made an impact as one item contained 7000 images. The presence of these items with these features cause the model to be biased toward detecting these banknotes with high accuracy and obtaining the rest of the objects with lower accuracy compared to the banknotes, as shown in table 4.3. Items with a red highlight have received high accuracy.

Table 4. 3 Illustrating AP of items

Object Name	AP)	Object Name	(AP)	Object Name	(AP)
1000 Dinar	94.609%	Clothing	72.831%	Remote Control	60.917%
10000 Dinar	87.057%	Coffee cup	72.326%	Rifle	27.487%
250 Dinar	99.015%	Electric Socket	92.090%	Shoe	52.993%
25000 Dinar	74.473%	Fan	90.220%	Shower	79.606%
500 Dinar	91.957%	Flipflops	78.189%	Sink	81.810%
5000 Dinar	97.881%	Glasses	74.451%	Sofa bed	80.737%
50000 Dinar	93.677%	Kettle	93.714%	Spoon	46.054%
Bed	58.176%	Key	56.820%	Stairs	84.617%
Book	55.351%	Knife	59.481%	Stick	44.130%
Bottle	88.573%	Laptop	81.091%	Table	83.672%
Cabinetry	63.972%	Medicinal Tab	34.263%	Television	37.747%
Car	62.590%	Mobile phone	81.025%	Toilet Seat	90.393%
Cat	83.702%	Mouse	77.414%	Wastebasket	82.887%
Chair	88.786%	Person	60.219%	Water Dispenser	76.198%
Charger	82.616%	Refrigerator	53.904%		

To compensate for this, the YOLOv3 multi-stage system has been presented. The proposed design is a YOLOv3 multi-stage system, which is constructed of two-stage detection and two YOLOv3 models. Each model is trained on a specific dataset. This design increases the detection accuracy of each stage.

The system's first stage is the YOLOv3 objects detection model, which is trained on a dataset containing 38 objects and achieved a mAP of 85.354% and 12.92 FPS. In the second stage of the system, two directions could be taken: Direction A, a direct detection and classification of 37 objects, or Direction B, which is used for Iraqi banknotes classification. When Iraqi banknotes are detected in the first stage, a split occurs and the banknotes are classified as: "250 Dinar, 500 Dinar, 1000 Dinar, 5000 Dinar, 10000 Dinar", 25000 Dinar, 50000 Dinar". The Iraqi banknotes detector achieved a mAP of 97.506% and 11.80 FPS Figure 4.4 shows the effect of using two-stage YOLOv3 detection on the AP distribution of objects.

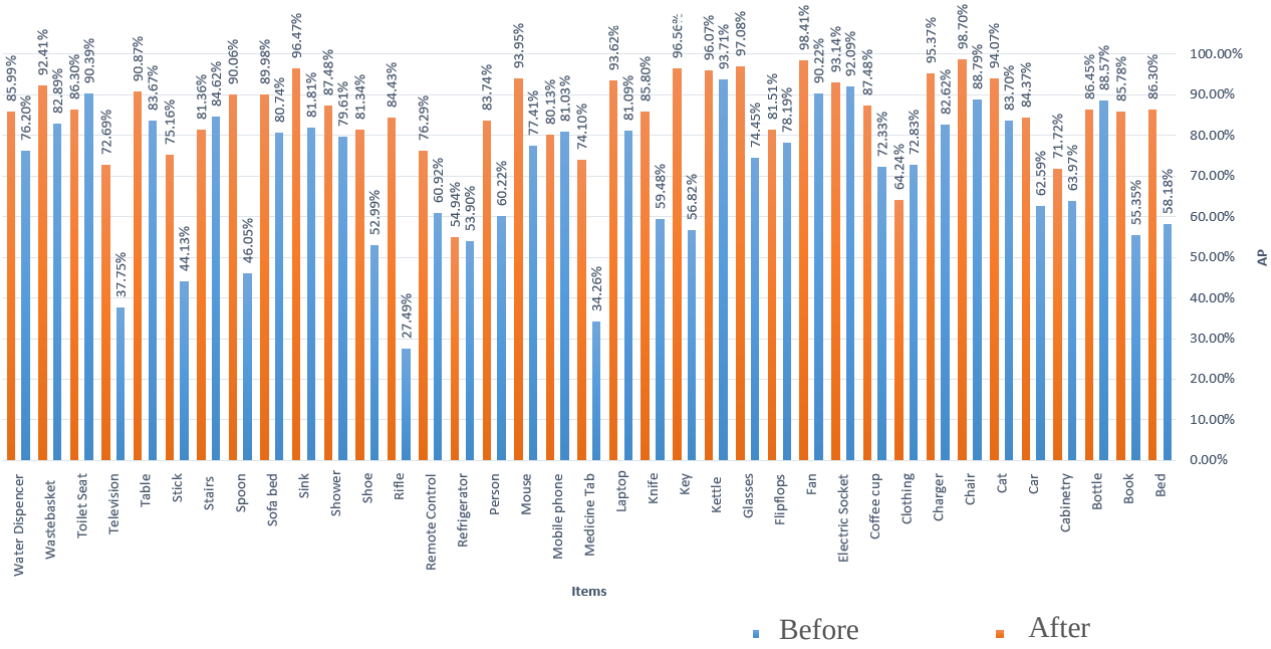


Figure 4.4 Objects' AP before and after using two-stage detection

4.3.5 Custom Anchors

The k-means clustering algorithm has been applied to datasets to get the suitable size of the anchor boxes, which are used during the prediction of a bounding box. Three anchor boxes are assigned to each detection scale, depending upon the size of the object in the dataset. The 9 anchor boxes generated by running the k-means clustering on each dataset are:

Model-1 (38 Objects): (21, 33), (76, 76), (59, 260), (199, 107), (160, 334), (123, 188), (381, 371), (270, 334), (302,201).

Model-2 (Banknotes): (218, 105), (142, 168), (283, 131), (139, 293), (215, 216), (331, 151), (176, 372), (390, 180), (273, 272).

Each model is trained with these custom anchors, and the mAP and evaluation time of each model are shown in table 4.4.

Table 4.4 Illustrating the mAP of models using costum anchors

Input Layer Size	Model-1 (38 Objects)		Model-2 (Banknotes)	
	mAP	FPs	mAP	FPs
416×416	86.886%	12.61	97.647%	11.01

Experiments indicate that there exists a direct relationship between the custom anchors used and mAP. Custom anchor boxes have improved the detection accuracy of each model. But the improvement has been very small because the anchors used in the first experiment were based on the Coco dataset, which is similar to the current dataset. But the main effect of using custom anchors has been to reduce the number of epochs that are needed to reach a learning study state from 100 epochs to almost 50 epochs.

4.4 Inference Mode

Along with the loss and mAP evaluation, the final system has been tested on new data from a live video camera and the results are seen visually as shown in figures 4.5 and 4.6. All the tested objects in all frames are correctly detected and classified.

Figure 4.5 Banknotes model inference mode

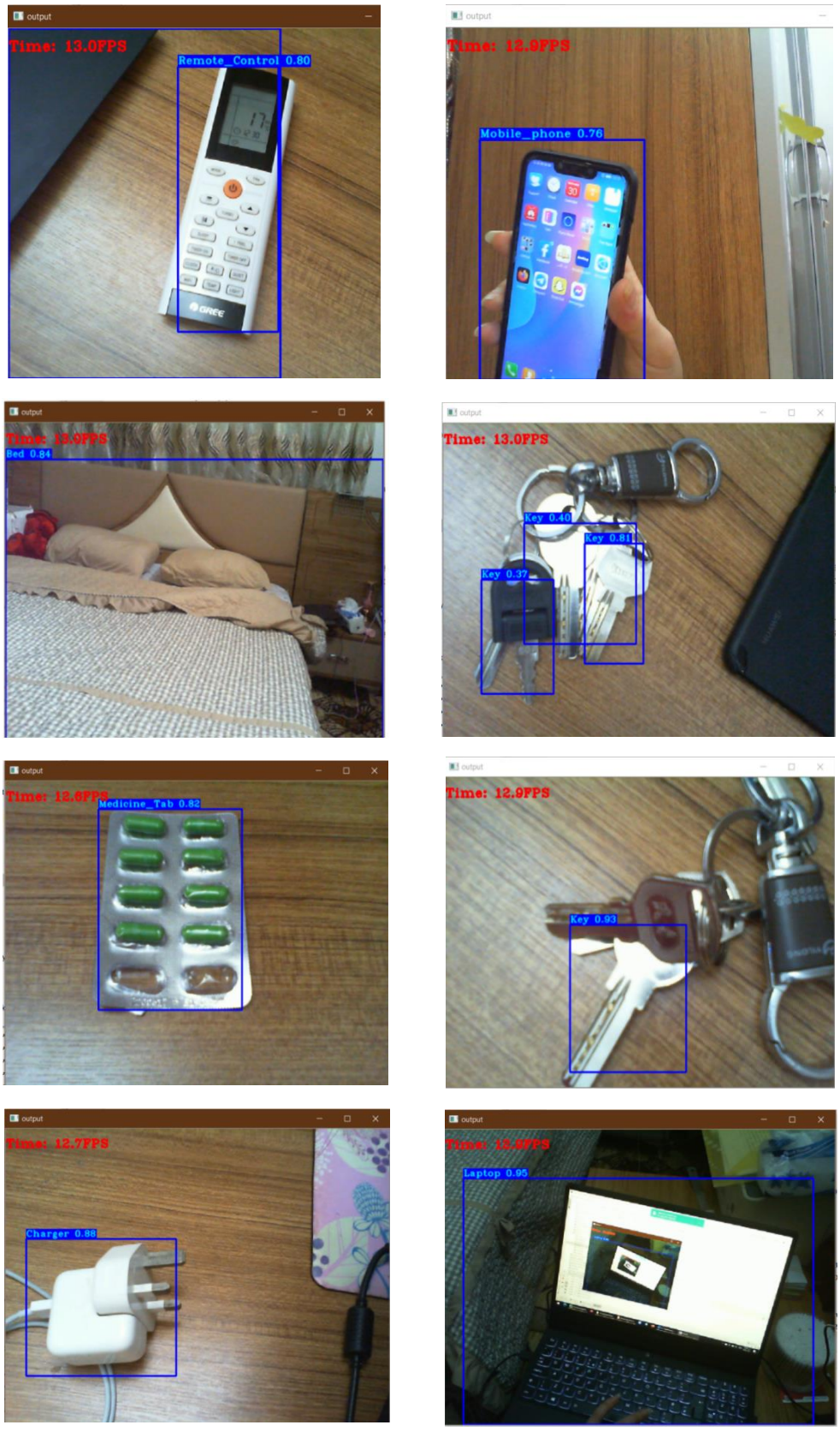


Figure 4.6 YOLOv3 model inference mode

4.5 Summary

Table 4.5 indicates the results of all the experiments discussed in this chapter. All these are results achieved by using 100 epochs except (38 Objects – Transfer learning, Custom anchors and 7 Iraqi Banknotes – Transfer learning, Custom anchors) are achieved using 50 epochs only.

Table 4. 5 Training results summary

Model	Input Image Size	mAP	FPS	Training time	Dataset	GPU Performance
44 Objects – Trained from scratch	320 × 320	4.267%	15.65	2 weeks	100%	High
44 Objects – Trained from scratch	416 × 416	8.564%	12.17	1 month	100%	Low
44 Objects – Trained from scratch	512 × 512	10.654%	10.85	2 weeks	100%	High
44 Objects – Transfer learning	416 × 416	73.403%	12.53	4 days	20%	High
38 Objects – Transfer learning	416 × 416	85.354%	12.92	3 days	20%	High
7 Iraqi Banknotes – Transfer learning	416 × 416	97.506%	11.80	1 days	20%	High
38 Objects – Transfer learning , Custom anchors	416 × 416	86.886%	12.61	36 Hr.	20%	High
7 Iraqi Banknotes – Transfer learning, Custom anchors	416 × 416	97.647%	11.01	12 Hr.	20%	High

4.6 Comparison with Other Assistance Systems

Although the comparison between the assistance devices proposed in this thesis and other assistive devices will not be accurate due to the difference in the hardware used, algorithms, the quantity and quality of the dataset. But it will help to know where this thesis has reached in general, and it indicates that the proposed algorithm gave good results. The proposed system was closer to that highlighted in red in table 4.6, which had an accuracy of 84.61% with 16 objects taken from the COCO Dataset, while the proposed assistance device in this thesis achieved an accuracy of up to 88.585% and 97.647 % for Iraqi banknotes, respectively. And detect 44 objects in a dataset that has been chosen using a completely patient-centered approach, which includes 44,000 images, 70% were collected manually. As shown in table 4.6. Which is highlighted in yellow.

Table 4.6 Comparison with Other Assistance Systems

Year	Author	No. of objects	Algorithm/ Devices	Dataset	Accuracy	FPS
2017	J. Zraqou et al. [13]	25	Matching using Laplacian/ Sensor	300 Images	76%	1
2018	Nishajith.A et al. [16]	90	SSD and MobileNet / Raspberry Pi3	COCO	21	/
2018	M. Ghilardi et al. [7]	pedestrian traffic lights sign	Yolo Full YOLO-Tiny	4,399 Images	76.2% 35.8%	/
2019	S. Shadi et al. [17]	15	DeepLabv3+ model	2760 Images	78%	23 - 30

2019	B. Kaur and J. Bhattacharya [18]	obstacles	CNN	PASCAL	/	/
2019	R. Ribani and M. Marengoni [19]	4	SSD with ResNet-50 / Raspberry Pi3	4 class COCO	22%	1
2019	A. Bhandari et al. [20]	obstacles	YOLO / Raspberry Pi3	COCO	/	/
2020	M. Afif et al. [21]	16	RetinaNet	8000 Images	84.61%	/
2020	L. Abraham et al. [8]	walking navigation	YOLO / Raspberry Pi4	COCO	/	/
2020	S. Shaikh [24]	/	YOLO	COCO	/	/
2021	Propose system	44	YOLOV3 two-stage models / Raspberry Pi3 and NVIDIA Jetson Nano	44000 Images	88.585% and 97.647% for Iraqi banknotes	13.11 5

CHAPTER FIVE

Hardware Implementation

5.1 Raspberry Pi 3 Implementation

The hardware used to implement YOLOv3 and perform objects detection on live video feeds from a USB webcam is a Raspberry Pi 3 (RPI) Model B, which is a single-board computer that comes with these specifications (see Appendix-D for more details):

- Quad Core 1.2 GHz Broadcom BCM2837 64 bit CPU.
- 1 GB RAM.
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board.
- 100 Base Ethernet.
- 4 USB-2 ports.
- Full size HDMI.
- CSI camera port for connecting a RPI camera.
- Micro SD port for loading your operating system and storing data.
- The switch has been upgraded to a Micro USB power source capable of delivering up to 2.5A.

Figure 5.1 shows the RPI 3 Model B which is used to implement the proposed system.

Figure 5.1 RPI 3 Model B

In order to install the required packages for running YOLOv3 on RPI, the RPI operating system image (.IMG) has been downloaded and flashed onto a micro-SD card. The RPI runs on a Linux kernel-based operating system that runs from the SD card, and it has no built-in memory other than the ROM. After that, a local network and wireless connectivity were set up.

Then all the needed packages and libraries to run the YOLOv3 objects detector on RPI have been installed, such as (pip3, OpenCV compatible with RPI, TF that satisfies the requirements of RPI, NumPy, SciPy, wget, seaborn, tqdm, pandas, awscli, and urllib3). Then the YOLOv3 trained model was loaded onto the SD card of the RPI.

The direct compilation of YOLOv3 on the RPI resulted an error message indicating limited power, and loading the weight files into the model took a long time and caused the CPU to run hot. Due to the high temperatures, it eventually shut down. The reason for this is that the RPI has a weak processor and limited RAM, so the RPI needs to use a model that consumes less processing power. The result of compilation YOLOv3 is shown in figure 5.2.

Figure 5.2 Error message of YOLOv3 direct compilation

5.2 Tiny-YOLOv3 Implementation

Tiny-YOLOv3 is a simplified version of YOLOv3, which has a much smaller number of convolution layers than YOLOv3. The Tiny-YOLOv3 algorithm is well suited for use on embedded platforms. Although the detection accuracy is lower than that of YOLOv3, this is due to Tiny-YOLOv3 reducing the YOLOv3 feature detection network darknet-53 to a 7-layer traditional convolution and a 6-layer max pooling, with a 13×13 and 26×26 scale prediction network used to predict the target [69].

Tiny-YOLOv3 has been implemented and trained in the same way as the YOLOv3 with 100% dataset. The results of training Tiny-YOLOv3 indicate that the detection accuracy of large networks of YOLOv3 is higher than Tiny-YOLOv3, as well, the evaluation time of Tiny-YOLOv3 is larger than the original YOLOv3 model, as shown in table 5.1.

Table 5.1 Illustrating the mAP of Tiny-YOLOv3 and YOLOv3 models

Type	Model-1 (38 Objects)		Model-2 (Banknotes)		Dataset
	mAP	FPs	mAP	FPs	
YOLOv3	86. 886%	12.61	97.647%	11.01	20%
Tiny-YOLOv3	77.700%	29.30	91.235%	29.59	100%

Tiny-YOLOv3 model is loaded on RPI and inference mode is performed to detect objects in each video frame from the USB camera but it still not fast enough which achieve 0.4 -1 FPS as shown in figure 5.3.

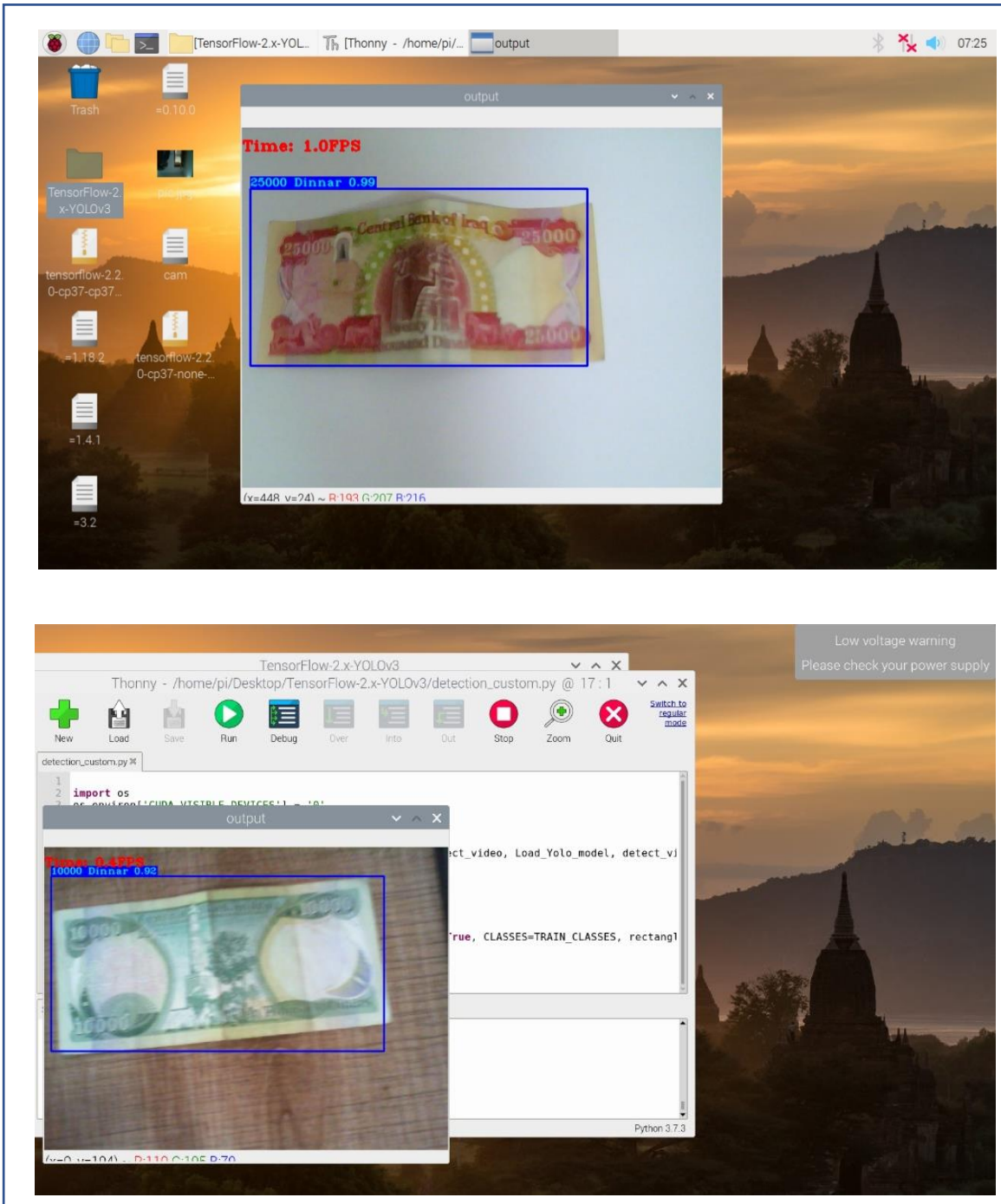


Figure 5.3 Running Tiny-YOLOv3 on RPI 3

5.3 NVIDIA Jetson Nano Implementation

The NVIDIA Jetson Nano is an NVIDIA product that can implement AI solutions with the power of GPU computation. The major characteristics of NVIDIA Jetson Nano are a low cost and great adaptability for running a wide range of deep networks, including full native versions of popular machine learning frameworks like TF, Caffe, PyTorch, Keras, and MXNet. These networks can be utilized to create autonomous machines and complex AI systems by incorporating powerful capabilities such as image recognition, object identification and localization, posture estimation, semantic segmentation, video enhancement, and intelligent analytics. The NVIDIA Jetson Nano contains a flash-based micro-SD card for non-volatile storage, and operates at ~10W. The specifications of the NVIDIA Jetson Nano used in the YOLOv3 implementation were (see Appendix-E for more details):

- CPU Quad-core ARM A57 running at 1.43 GHz.
- Maxwell GPU with 128 cores.
- 4 GB 64-bit LPDDR 25.6 GB/s memory
- HDMI and display port output.
- Micro SD card storage (not included).
- 4K video encoding.
- 4K video decoding.
- USB-C 5V/3A.
- USB 4x USB 3.0, USB 2.0 Micro-B.
- Connectivity Gigabit Ethernet, M.2 Key E.

Figure 5.4 shows the NVIDIA Jetson Nano which is used to implement the proposed system.

Figure 5.4 NVIDIA Jetson Nano Kit

The first boot of the NVIDIA Jetson Nano needs four things: a micro-SD card (minimum of 32 GB), a 5V 2.5A Micro USB power supply, an Ethernet cable, and the JetPack 4.5 NVIDIA Jetson Nano operation (.IMG) file flashed to the micro-SD card.

In order to configure the NVIDIA Jetson Nano to run the YOLOv3 model, the system package, NumPy, Keras, TF, and the Jetson Inference engine have been installed. Then the python environment has been configured using python virtual environments to keep python development environments independent and separate from each other and to avoid having to maintain a micro-SD for each development environment used on the NVIDIA Jetson Nano. The python virtual environment is managed using *virtualenv* and *virtualenvwrapper* which were installed before.

NVIDIA has provided an official release of TF for the NVIDIA Jetson Nano. However, the most recent release of TF3.4 is incompatible with the current system version (JetPack4.5), so TF2.3 has to be manually loaded.

The Tiny-YOLOv3 model is loaded on the NVIDIA Jetson Nano and detection is performed on each video frame from the USB camera, which achieves 4-5 FPS, as shown in figure 5.5.

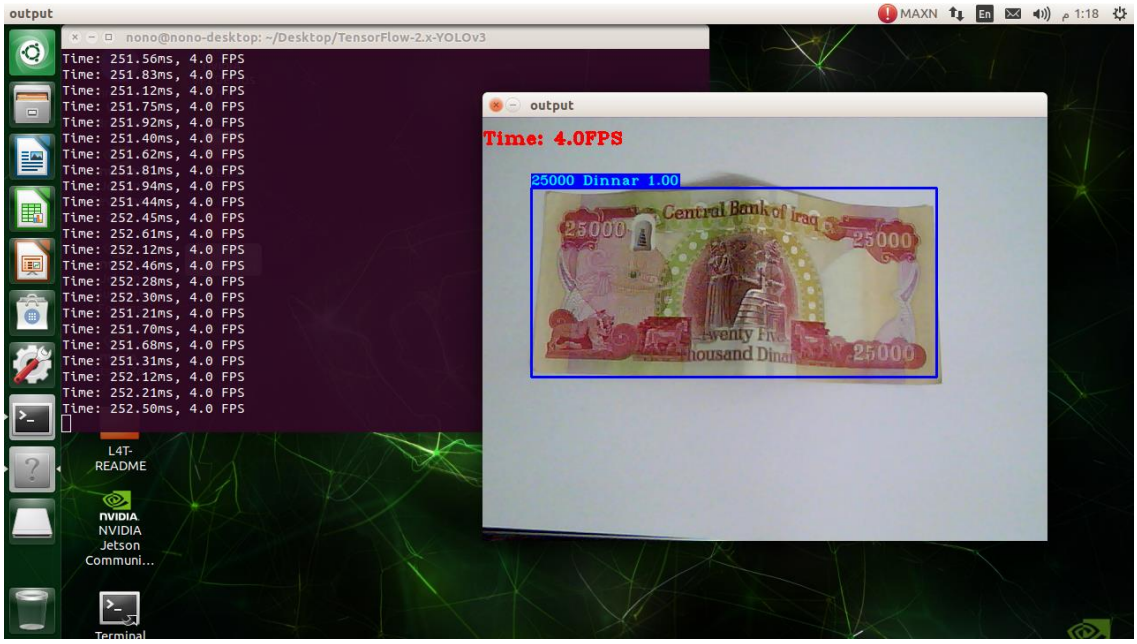


Figure 5.5 Running Tiny-YOLOv3 on NVIDIA Jetson Nano

5.4 NVIDIA Jetson Nano vs. Raspberry Pi 3

Table 5.2 depicts the inference speed measured in FPS, on different hardware. It shows that the implementation of the proposed model on Raspberry Pi 3B and NVIDIA Jetson Nano produced DNR (did not run) results, which occurred due to limited memory capacity. Therefore, Tiny-YOLOv3 for the proposed model was used on the Raspberry Pi 3B and NVIDIA Jetson Nano.

The results of the implementation show that the inference speed of the NVIDIA Jetson Nano is significantly faster than the Raspberry Pi 3B. As well, the speed of PC GPU implementation is higher than others.

Table 5.2 Inference time on different hardware

YOLOv3 Type	Pc GPU	Raspberry Pi 3B	NVIDIA Jetson Nano
YOLOv3	13.1 FPS	DNR	DNR
Tiny-YOLOv3	31 FPS	1 FPS	5 FPS

CHAPTER SIX

Conclusions and Future Works

6.1 Conclusions

The current thesis presents a visual impaired assistance system based on deep learning and using a multi-stage YOLOv3 network architecture with a total accuracy of 88.585% and 97.647 %for Iraqi banknotes. The final developed model has been implemented on the Raspberry Pi 3B and the NVIDIA Jetson Nano. The most important conclusions can be summarized as follows:

- The increase in the input layer size of the YOLOv3 network is directly proportional to detector accuracy and inversely proportional to detector speed.
- The results show that transfer learning yields an increase in the accuracy of the YOLOv3 objects detector from 8.564% mAP to 73.403%. The time required in the deep learning process to produce an accurate result can be reduced by the careful use of transfer learning instead of training the model from scratch with random initial weight.
- The adaptation of the learning rate of the YOLOv3 model is based on cosine decay with warm-up and ADAM optimizer produced robust results.
- GIoU enhances the distance measurement between the real box and the prediction box.
- The Multi-stage YOLOv3 model solves the problem of training distinct types of objects by enhancing the learning process.

- The main effect of using custom anchors is to reduce the number of epochs that are needed to reach a learning study state from 100 epochs to almost 50 epochs.
- The implementations of the two embedded system platforms show that the detection speed of the NVIDIA Jetson Nano (5 FPS) is faster than the Raspberry Pi 3B (1 FPS).
- The proposed system is designed in such a way to be general that it can be extended to use it in different field of object detection.

6.2 Future Works

Some suggestions for future works can be summarized as follows:

- Using Tensor RT to increase objects detection speed on GPU.
- Adding a color detection model.
- Adding a detection model to recognize the members of the blind person family

References

[1] A. Parab, R. Nagare, O. Kolambekar, and P. Patil, “Electronic Orientation Aid for Visually Impaired using Graphics Processing Unit (GPU),” *ITM Web Conf.*, vol. 32, p. 03054, 2020, doi: 10.1051/itmconf/20203203054.

[2] R. Rajwani, D. Purswani, and P. Kalinani, “Proposed System on Object Detection for Visually Impaired People,” *Int. J. Inf. Technol.*, vol. 4, no. 1, pp. 1–6, 2018.

[3] S. Vaidya, N. Shah, N. Shah, and R. Shankarmani, “Real-Time Object Detection for Visually Challenged People,” *Proc. Int. Conf. Intell. Comput. Control Syst. ICICCS 2020*, no. Iciccs, pp. 311–316, 2020, doi: 10.1109/ICICCS48265.2020.9121085.

[4] “WHO | World Health Organization.” https://www.who.int/en (accessed Oct. 08, 2020).

[5] H. Jabnoun, F. Benzarti, and H. Amiri, “Object detection and identification for blind people in video scene,” in *International Conference on Intelligent Systems Design and Applications, ISDA*, Jun. 2016, vol. 2016-June, pp. 363–367, doi: 10.1109/ISDA.2015.7489256.

[6] R. C. Joshi, S. Yadav, and M. K. Dutta, “YOLO-v3 Based Currency Detection and Recognition System for Visually Impaired Persons,” *2020 Int. Conf. Contemp. Comput. Appl. IC3A 2020*, pp. 280–285, 2020, doi: 10.1109/IC3A48958.2020.233314.

[7] M. C. Ghilardi, G. Simoes, J. Wehrmann, I. H. Manssour, and R. C. Barros, “Real-Time Detection of Pedestrian Traffic Lights for Visually-Impaired People,” *Proc. Int. Jt. Conf. Neural Networks*, vol.

2018-July, pp. 1–8, 2018, doi: 10.1109/IJCNN.2018.8489516.

[8] L. Abraham, N. S. Mathew, L. George, and S. S. Sajan, “VISION-Wearable Speech Based Feedback System for the Visually Impaired using Computer Vision,” *Proc. 4th Int. Conf. Trends Electron. Informatics, ICOEI 2020*, no. Icoei, pp. 972–976, 2020, doi: 10.1109/ICOEI48184.2020.9142984.

[9] O. Masurekar, O. Jadhav, P. Kulkarni, and S. Patil, “Real Time Object Detection Using YOLOv3,” *Int. Res. J. Eng. Technol.*, vol. 07, no. 03, pp. 3764–3768, 2020.

[10] B. Schauerte, M. Martinez, A. Constantinescu, and R. Stiefelhagen, “An assistive vision system for the blind that helps find lost things,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7383 LNCS, no. PART 2, pp. 566–572, 2012, doi: 10.1007/978-3-642-31534-3_83.

[11] M. R. Nalawade, V. Wagh, and S. Kamble, “An Approach for Object and Scene Detection for Blind Peoples Using Vocal Vision .,” vol. 4, no. 12, pp. 1–3, 2014.

[12] P. S. P. Jadhav, S. Tomy, S. S. Jayswal, H. D. Dhaware, and A. R. Vijapure, “Object Detection in Android Smartphone for Visually Impaired Users,” *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 5, no. 11, pp. 332–334, 2016, doi: 10.17148/IJARCCE.2016.51171.

[13] J. S. Zraqou, W. M. Alkhadour, and M. Z. Siam, “Real-Time Objects Recognition Approach for Assisting Blind People,” *Int. J. Curr. Eng. Technol.*, vol. 7, no. 1, pp. 120–125, 2017.

[14] Pachhaiammal@Priya M, “SMART NAVIGATION TO ASSIST BLIND PEOPLE FOR OBJECT,” vol. 5, no. 4, pp. 2682–2687,

2017.

[15] Govardhan.S.D, Kumar.G, Mariyappan.S, N. Kumar.G, and N. Asir.J, “SMART OBJECT DETECTOR FOR VISUALLY IMPAIRED,” *Int. J. Recent Trends Eng. Res.*, no. March-2017 [ISSN: 2455-1457] DOI : 10.23883/IJRTER.CONF.20170331.038.5C8TI, pp. 192–196, 2017.

[16] Nishajith.A, Nivedha.J, Shilpa.S.Nair, and P. M. Shaffi.J, “Smart Cap – Wearable Visual,” *2018 Int. Conf. Inven. Res. Comput. Appl.*, no. Icirca, ISBN:978-1-5386-2456-2, pp. 275–278, doi: 10.1109/ICIRCA.2018.8597327., 2018.

[17] S. Saleh, H. Saleh, and W. Hardt, “Outdoor navigation for visually impaired based on deep learning,” *CEUR Workshop Proc.*, vol. 2514, no. November, pp. 397–406, 2019.

[18] B. Kaur and J. Bhattacharya, “Scene perception system for visually impaired based on object detection and classification using multimodal deep convolutional neural network,” *J. Electron. Imaging*, vol. 28, no. 01, p. 1, 2019, doi: 10.1117/1.jei.28.1.013031.

[19] R. Ribani and M. Marengoni, “Vision substitution with object detection and vibrotactile stimulus,” *VISIGRAPP 2019 - Proc. 14th Int. Jt. Conf. Comput. Vision, Imaging Comput. Graph. Theory Appl.*, vol. 4, no. Visigrapp, pp. 584–590, 2019, doi: 10.5220/0007577205840590.

[20] A. Bhandari, R. Gorad, S. Thakur, and J. Sangoi, “Charanatra : A smart assistive footwear for visually impaired,” *Int. J. Adv. Res. Ideas Innov. Techn*, vol. 5, no. 2, pp. 850–852, 2019.

[21] M. Afif, R. Ayachi, Y. Said, E. Pissaloux, and M. Atri, “An

Evaluation of RetinaNet on Indoor Object Detection for Blind and Visually Impaired Persons Assistance Navigation," *Neural Process. Lett.*, vol. 51, no. 3, pp. 2265–2279, 2020, doi: 10.1007/s11063-020-10197-9.

[22] A. Salem and B. Sama, "Enhancement and Development of Smart Glasses System for Visually Impaired Persons by Using Intelligent System," *Int. J. Comput. Sci. Mob. Comput.*, vol. 9, no. 8, pp. 40–49, 2020.

[23] S. Yadav, Z. A. Ansari, and K. G. Singh, "CURRENCY DETECTION FOR VISUALLY," *Int. J. Emerg. Technol. Innov. Res. www.jetir.org | UGC issn Approv. ISSN2349-5162*, vol. 7, no. 5, pp. 999–1002, 2020.

[24] Shifa Shaikh, "Assistive Object Recognition System for Visually Impaired," *Int. J. Eng. Res. Technol. ISSN 2278-0181*, vol. V9, no. 09, pp. 736–740, 2020, doi: 10.17577/ijertv9is090382.

[25] J. F. Peters, "Foundations of Computer Vision," *B. Intell. Syst. Ref. Libr. ·*, vol. 124, no. March 2017, p. 431, 2017, doi: 10.1007/978-3-319-52483-2.

[26] L. Jiao *et al.*, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, no. 3, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.

[27] A. Sobti, C. Arora, and M. Balakrishnan, "Object Detection in Real-Time Systems: Going beyond Precision," *Proc. - 2018 IEEE Winter Conf. Appl. Comput. Vision, WACV 2018*, vol. 2018-Janua, pp. 1020–1028, 2018, doi: 10.1109/WACV.2018.00117.

[28] P. F. Felzenszwalb, R. B. Girshick, D. Mcallester, and D. Ramanan,

"Object Detection With Partbase," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010.

[29] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[31] R. Rai, S. Shukla, and B. Singh, "Histograms of Oriented Gradients for Human Detection," *IEEE Trans. Ind. Informatics*, vol. 16, no. 7, pp. 4714–4725, 2005, doi: 10.1109/TII.2019.2950094.

[32] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," *IEEE Int. Conf. Image Process.*, vol. 1, pp. 900–903, 2002, doi: 10.1109/icip.2002.1038171.

[33] N. H. Farhat, "Support-Vector Networks CORINNA," *IEEE Expert. Syst. their Appl.*, vol. 7, no. 5, pp. 63–72, 1995, doi: 10.1109/64.163674.

[34] R. Barmaki, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting* Yoav," *J. Comput. Syst. Sci. 55, 119?139 Artic.*, vol. 139, pp. 651–655, 1996, doi: 10.1145/2818346.2823306.

[35] D. M. and D. R. P. F. Felzenszwalb, R. B. Girshick, "Object Detection with Discriminatively Trained," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 2, pp. 1627–1645, doi: 10.1109/TPAMI.2009.167., 2010.

[36] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, Sep. 2014, doi: 10.1109/CVPR.2014.81.

[37] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[38] C. G. Pachón, D. M. Ballesteros, and D. Renza, “Fake banknote recognition using deep learning,” *Appl. Sci.*, vol. 11, no. 3, pp. 1–20, 2021, doi: 10.3390/app11031281.

[39] M. Z. Alom *et al.*, “A state-of-the-art survey on deep learning theory and architectures,” *Electron.*, vol. 8, no. 3, 2019, doi: 10.3390/electronics8030292.

[40] C. Borngrund, Machine vision for automation of earth-moving machines : Transfer learning experiments with YOLOv3,” *Luleå Univ. Technol. Dep. Comput. Sci. Electr. Sp. Eng.*, p. Retrieved from http://urn.kb.se/resolve?urn=urn:nb, 2019.

[41] J. N. Slettevold, “Deep learning in Dynamic Imager,” *Master’s Thesis; NTNU*, no. June, 2018, [Online]. Available: http://hdl.handle.net/11250/2566506.

[42] M. Lin, Q. Chen, and S. Yan, “Network In Network,” *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, Dec. 2013, Accessed: Aug. 14, 2021. [Online]. Available: https://arxiv.org/abs/1312.4400v3.

[43] V. N. Nguyen, “Advancing Deep Learning for Automatic Autonomous Vision-based Power Line Inspection,” *IEEE Power Energy Technol. Syst. J.*, vol. 6, no. 1, 2019.

[44] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[45] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[46] S. R. K. H. R. G. J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with,” *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1 NIPS'15, pp. 91–99, 2015, doi: 10.2307/j.ctt1d98bxx.10.

[47] W. Liu *et al.*, “SSD: Single shot multibox detector,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.

[48] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[49] A. F. Joseph Redmon∗, Santosh Divvala∗, Ross Girshick, “You Only Look Once: Unified, Real-Time Object Detection Joseph,” *J. Chem. Eng. Data*, vol. 27, no. 3, pp. 779–788, 2016, doi: 10.1021/je00029a022.

[50] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[51] J. Redmon and A. Farhadi, “YOLO v.3,” *Tech Rep.*, pp. 1–6, 2018, [Online]. Available:

https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[52] Z. Deng, R. Yang, R. Lan, Z. Liu, and X. Luo, “SE-IYOLOV3: An accurate small scale face detector for outdoor security,” *Mathematics*, vol. 8, no. 1, pp. 1–12, 2020, doi: 10.3390/math8010093.

[53] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[54] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.

[55] R. Ribani and M. Marengoni, “A Survey of Transfer Learning for Convolutional Neural Networks,” *Proc. - 32nd Conf. Graph. Patterns Images Tutorials, SIBGRAPI-T 2019*, pp. 47–57, 2019, doi: 10.1109/SIBGRAPI-T.2019.00010.

[56] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11141 LNCS, pp. 270–279, 2018, doi: 10.1007/978-3-030-01424-7_27.

[57] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, “Generalized intersection over union: A metric and a loss for bounding box regression,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 658–666, 2019, doi: 10.1109/CVPR.2019.00075.

[58] R. Padilla, S. L. Netto, and E. A. B. Da Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” *Int. Conf. Syst. Signals, Image Process.*, vol. 2020-July, pp. 237–242, 2020, doi: 10.1109/IWSSIP48289.2020.9145130.

[59] H. Choi, H. Lee, H. You, S. Rhee, and W. Jeon, “and Error Matrix for Vegetation Cover Classification Assessment,” *Sensors Mater.*, vol. 31, no. 11, pp. 3849–3858, 2019.

[60] P. Henderson and V. Ferrari, “End-to-end training of object class detectors for mean average precision,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10115 LNCS, pp. 198–213, 2017, doi: 10.1007/978-3-319-54193-8_13.

[61] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” *Proc. - Int. Conf. Pattern Recognit.*, vol. 3, no. January 2006, pp. 850–855, 2006, doi: 10.1109/ICPR.2006.479.

[62] “Anchor Boxes for Object Detection - MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html (accessed Jun. 12, 2021).

[63] A. Kuznetsova *et al.*, “The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale,” *Int. J. Comput. Vis.*, vol. 128, no. 7, pp. 1956–1981, Nov. 2018, doi: 10.1007/s11263-020-01316-z.

[64] S. D. Al-Sheekh and M. D. Younus, “Real-Time Pose Estimation for Human-Robot Interaction,” in *2020 2nd Annual International Conference on Information and Sciences (AiCIS)*, Nov. 2020, pp. 86–90, doi: 10.1109/AiCIS51645.2020.00023.

[65] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of tricks for image classification with convolutional neural networks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 558–567, 2019, doi: 10.1109/CVPR.2019.00065.

[66] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of Freebies for Training Object Detection Neural Networks,” 2019, [Online]. Available: http://arxiv.org/abs/1902.04103.

[67] A. Ammar, A. Koubaa, M. Ahmed, A. Saad, and B. Benjdira, “Vehicle detection from aerial images using deep learning: A comparative study,” *Electron.*, vol. 10, no. 7, pp. 1–31, 2021, doi: 10.3390/electronics10070820.

[68] Z. Wang, “SEG-YOLO: Real-Time Instance Segmentation Using YOLOv3 and Fully Convolutional Network,” 2019, [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-271652.

[69] X. Gong, L. Ma, and H. Ouyang, “An improved method of Tiny YOLOV3,” *IOP Conf. Ser. Earth Environ. Sci.*, vol. 440, no. 5, 2020, doi: 10.1088/1755-1315/440/5/052025.

Appendix-A

Survey Form

إستطلاع رأي

العمر : ○ ضعيف البصر

الجنس : ○ فاقد البصر

إستطلاع رأي لمعرفة مدى اهمية كل اداة من الادوات الموضحة في الجدول ادناه لكل من المكفوفين او ضعاف البصر ، الهدف من الاستطلاع معرفة اهم الادوات لغرض انشاء نظام مساعدة ضعاف البصر.

ملاحظة: بإمكان المشاركين في الاستطلاع إضافة أدوات لم يتم ذكرها في الجدول.

ت	الادوات / Items	مهم جداً	مهم	نادر الأهمية	غير مهم
1	250 Dinar / 250 دينار				
2	500 Dinar / 500 دينار				
3	1000 Dinar / 1000 دينار				
4	5000 Dinar / 5000 دينار				
5	10000 Dinar / 10000 دينار				
6	25000 Dinar / 25000 دينار				
7	50000 Dinar / 50000 دينار				
8	Bed / سرير				
9	Book / كتاب				
10	Bottle / قنينة ماء				
11	Building / بناية				
12	Bus / حافلة				
13	Cabinetry / خزانة				
14	Camera / كاميرا				
15	Car / سيارة				
16	Cat / قطة				
17	Chair / كرسي				
18	Charger / شاحنة موبايل				
19	Clothing / ملابس				
20	Coffee Cup / كوب				
21	Computer keyboard / لوحة مفاتيح الحاسبة				
22	Dog / كلب				
23	Electric Socket / مقبس كهرباء				
24	Fan / مروحة				
25	Fire / نار				

				مطفئة حريق / Fire Extinguisher	26
				نعال- شبشب / Flipflops	27
				طعام / Food	28
				شوكة / Fork	29
				فاكهة / Fruit	30
				طباخ / Gas stove	31
				مناظر / Glasses	32
				منزل / House	33
				قوري / Kettle	34
				مفتاح / Key	35
				سكين / Knife	36
				لابتوب / Laptop	37
				دواء / Medicine Tab	38
				موبايل / Mobile phone	39
				ماوس حاسبة / Mouse	40
				قلم / Pen	41
				شخص / Person	42
				ثلاجة / Refrigerator	43
				جهاز تحكم - ريموت / Remote Control	44
				مقص / Scissors	45
				حذاء / Shoe	46
				دوش / shower	47
				مغسل / Sink	48
				قنفة / Sofa Bed	49
				ملعقة / Spoon	50
				سلالم- درج / Stairs	51
				عصا / Stick	52
				منضدة / Table	53
				تلفاز / Television	54
				مقعد تواليت / Toilet Seat	55
				ترفك لايت / Traffic light	56
				خضراوات / Vegetable	57
				سلة مهملات / Wastebasket	58
				ساعة / Watch	59
				ثاجة ماء / Water Dispenser	60
				سلاح / Weapon	61
				نافذة / Window	62

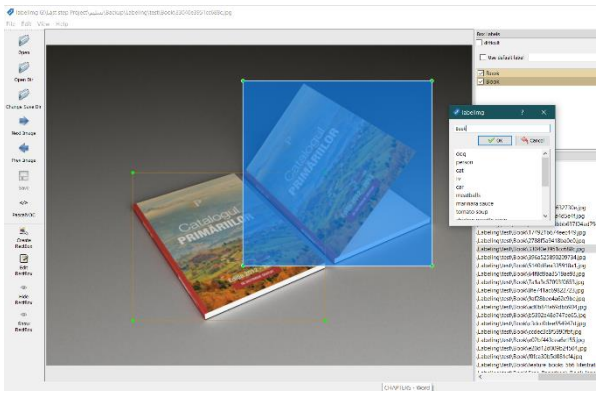
Survey Voting Results

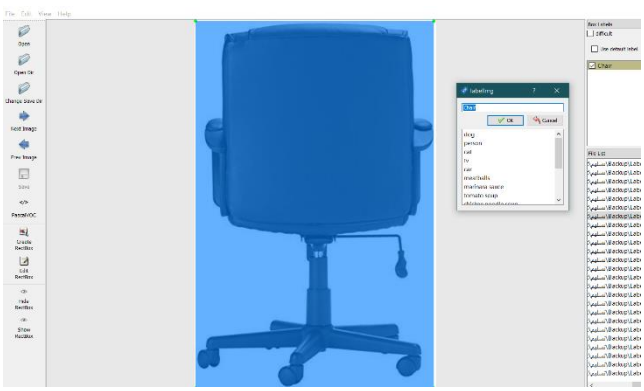
No	Items	Most important	important	Rarely important	not important
1	250 Dinar	15	3	1	1
2	500 Dinar	18	2	0	0
3	1000 Dinar	18	2	0	0
4	5000 Dinar	17	2	1	0
5	10000 Dinar	17	2	1	0
6	25000 Dinar	15	3	1	1
7	50000 Dinar	15	3	1	1
8	Bed	7	10	2	1
9	Book	0	11	6	3
10	Bottle	7	10	3	0
11	Building	0	2	2	16
12	Bus	0	4	11	5
13	Cabinetry	6	9	5	0
14	Camera	0	0	0	20
15	Car	15	3	1	1
16	Cat	0	8	7	5
17	Chair	18	2	0	0
18	Charger	20	0	0	0
19	Clothing	17	3	0	0
20	Coffee Cup	20	0	0	0
21	Computer keyboard	0	0	0	20
22	Dog	0	0	13	7
23	Electric Socket	17	2	1	0
24	Fan	3	12	4	1
25	Fire	2	2	7	9
26	Fire Extinguisher	0	0	0	20
27	Flipflops	18	2	0	0
28	Food	0	1	13	6
29	Fork	0	3	14	3
30	Fruit	1	0	3	16
31	Gas stove	1	3	9	7
32	Glasses	3	11	6	0
33	House	0	0	3	17

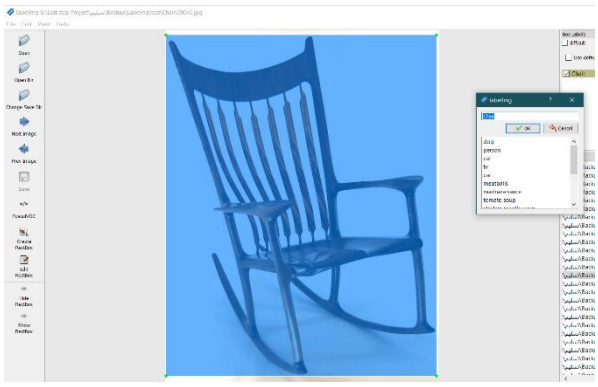
34	Kettle	7	9	4	0
35	Key	20	0	0	0
36	Knife	3	13	4	0
37	Laptop	7	10	1	2
38	Medicine Tab	19	1	0	0
39	Mobile phone	20	0	0	0
40	Mouse	5	12	1	2
41	Pen	0	0	2	18
42	Person	10	7	3	0
43	Refrigerator	2	9	6	3
44	Remote Control	10	9	1	0
45	Scissors	0	2	11	7
46	Shoe	16	4	0	0
47	shower	4	9	7	0
48	Sink	10	10	0	0
49	Sofa Bed	7	9	4	0
50	Spoon	11	7	2	0
51	Stairs	15	5	0	0
52	Stick	12	7	1	0
53	Table	8	9	2	1
54	Television	13	7	0	0
55	Toilet Seat	10	10	0	0
56	Traffic light	0	0	0	20
57	Vegetable	0	0	0	20
58	Wastebasket	17	3	0	0
59	Watch	0	0	18	2
60	Water Dispenser	6	14	0	0
61	Weapon	1	10	5	4
62	Window	0	1	11	8

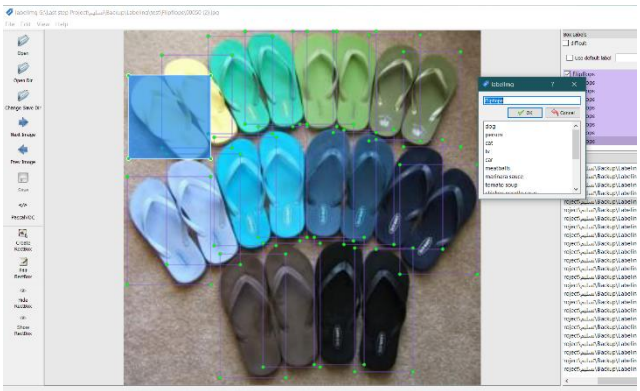
Appendix-B

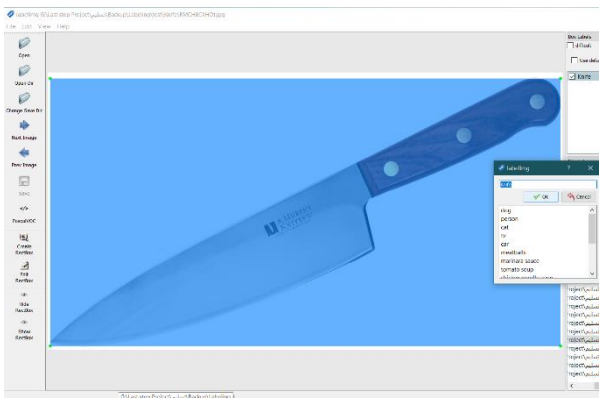
This examples illustrates the labeling process of images using the "LabelImg" tool.

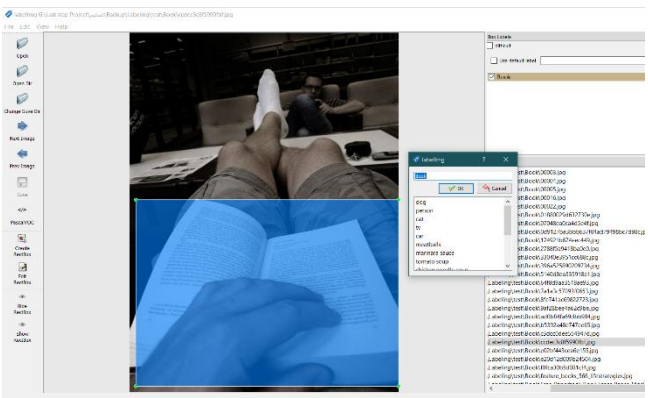












Appendix-C

Code

```
================== Darknet-53 coding ==================
def dark53(indata):
    indata = convol(indata, (3, 3, 3, 32))
    indata = convol(indata, (3, 3, 32, 64), dnsamp=True)
    for i in range(1):
         indata = resid(indata, 64, 32, 64)
    indata = convol(indata, (3, 3, 64, 128), dnsamp=True)
    for i in range(2):
        indata = resid(indata, 128, 64, 128)
    indata = convol(indata, (3, 3, 128, 256), dnsamp=True)
    for i in range(8):
        indata = resid(indata, 256, 128, 256)
    route_1 = indata
    I = convol(indata, (3, 3, 256, 512), dnsamp=True)
    for i in range(8):
        indata = resid(indata, 512, 256, 512)
    route_2 = indata
    I = convol(indata, (3, 3, 512, 1024), dnsamp=True)
    for i in range(4):
        indata = resid(indata, 1024, 512, 1024)
    return route_1, route_2, indata
================= YOLO layers coding ==================
def YOLOv3(in_lay, N_CLSS):
     route_1, route_2, convl = dark53(in_lay)
    convl = convol(convl, (1, 1, 1024, 512))
    convl = convol(convl, (3, 3, 512, 1024))
```

```
    convl = convol(convl, (1, 1, 1024, 512))
    convl = convol(convl, (3, 3, 512, 1024))
    convl = convol(convl, (1, 1, 1024, 512))
    conv_lobj_branch = convol(convl, (3, 3, 512, 1024))
    convlbox = convol(conv_lobj_branch, (1, 1, 1024, 3 * (N_CLSS + 5)),
activate=False, bn=False)
    convl = convol(convl, (1, 1, 512, 256))
    convl = upsample(convl)
    convl = tf.concat([convl, route_2], axis=-1)
    convl = convol(convl, (1, 1, 768, 256))
    convl = convol(convl, (3, 3, 256, 512))
    convl = convol(convl, (1, 1, 512, 256))
    convl = convol(convl, (3, 3, 256, 512))
    convl = convol(convl, (1, 1, 512, 256))
    conv_mobj_branch = convol(convl, (3, 3, 256, 512))
    convmbox = convol(conv_mobj_branch, (1, 1, 512, 3 * (N_CLSS +
5)), activate=False, bn=False)
    convl = convol(convl, (1, 1, 256, 128))
    convl = upsample(convl)
    convl = tf.concat([convl, route_1], axis=-1)
    convl = convol(convl, (1, 1, 384, 128))
    convl = convol(convl, (3, 3, 128, 256))
    convl = convol(convl, (1, 1, 256, 128))
    convl = convol(convl, (3, 3, 128, 256))
    convl = convol(convl, (1, 1, 256, 128))
    conv_sobj_branch = convol(convl, (3, 3, 128, 256))
    convsbox = convol(conv_sobj_branch, (1, 1, 256, 3 * (N_CLSS + 5)),
activate=False, bn=False)
    return [convsbox, convmbox, convlbox]
```

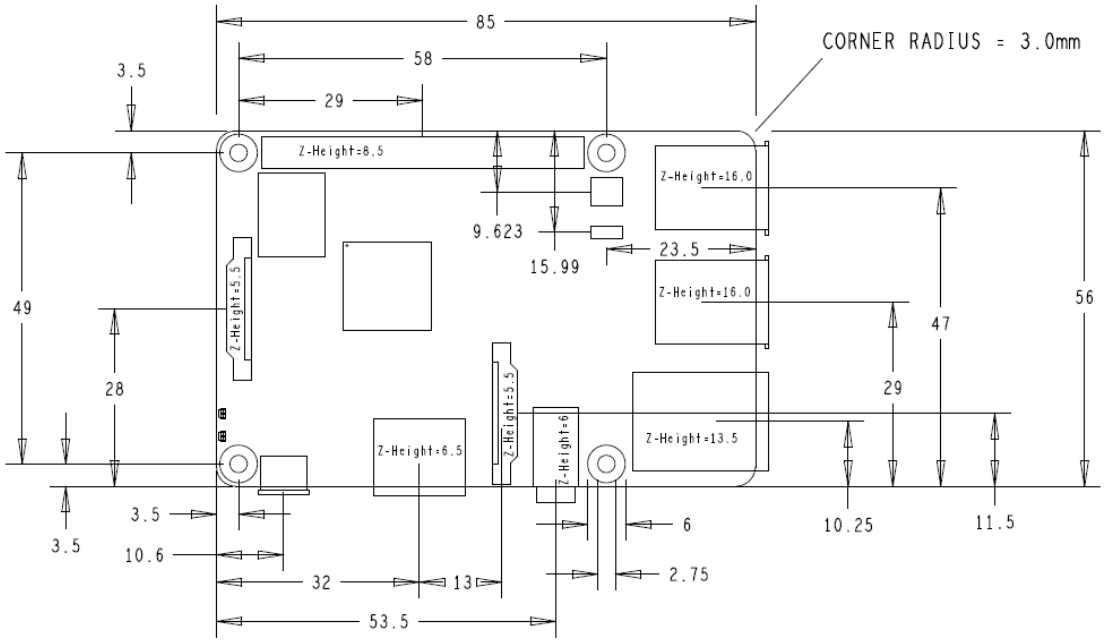
================ decode function coding ================

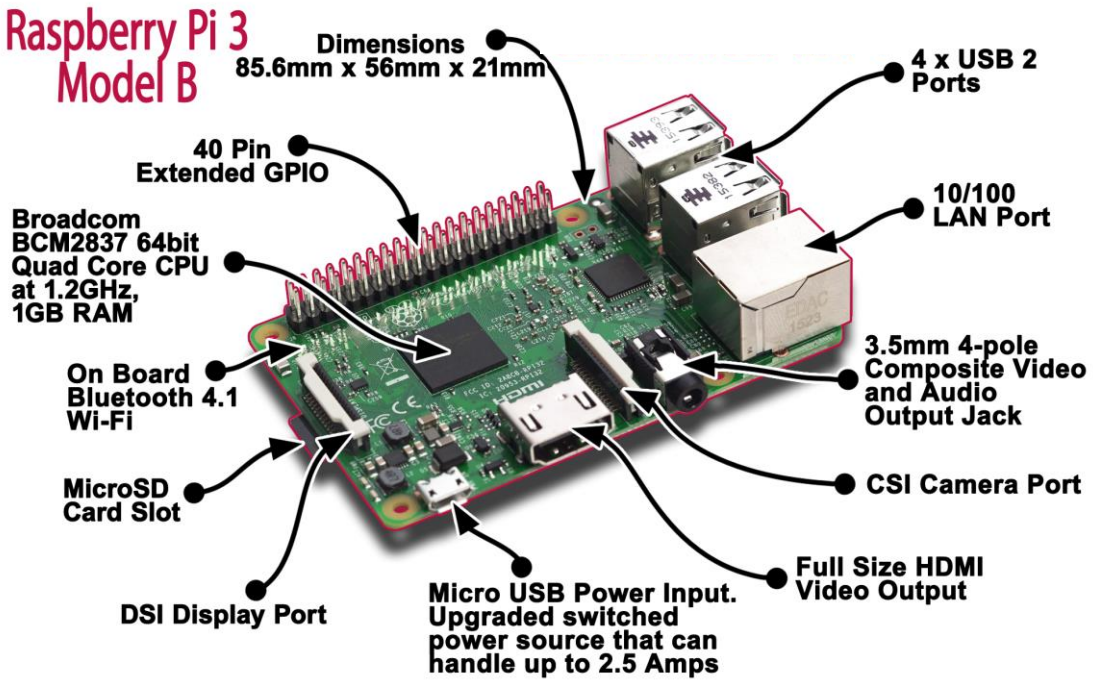
```
def decode(convout, N_CLSS, i=0):
    conv_shape = tf.shape(convout)
    batch_size = conv_shape[0]
    outsize = conv_shape[1]
    convout = tf.reshape(convout, (batch_size, outsize, outsize, 3, 5 +
N_CLSS))
    conv_raw_dxdy = convout[:, :, :, :, 0:2]
    conv_raw_dwdh = convout[:, :, :, :, 2:4]
    conv_raw_conf = convout[:, :, :, :, 4:5]
    conv_raw_prob = convout[:, :, :, :, 5:]
    y = tf.range(outsize, dtype=tf.int32)
    y = tf.expand_dims(y, -1)
    y = tf.tile(y, [1, outsize])
    x = tf.range(outsize, dtype=tf.int32)
    x = tf.expand_dims(x, 0)
    x = tf.tile(x, [outsize, 1])
    xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :, tf.newaxis]], axis=-1)
    xy_grid = tf.tile(xy_grid[tf.newaxis, :, :, tf.newaxis, :], [batch_size, 1,
1, 3, 1])
    xy_grid = tf.cast(xy_grid, tf.float32)
    pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) * ST[i]
    pred_wh = (tf.exp(conv_raw_dwdh) * ANCHORS[i]) * ST[i]
    pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
    pred_conf = tf.sigmoid(conv_raw_conf)
    pred_prob = tf.sigmoid(conv_raw_prob)
    return tf.concat([pred_xywh, pred_conf, pred_prob], axis=-1)
```

Appendix-D

Raspberry Pi 3 Model B Datasheet

Physical Specifications





Warnings

- This product should be operated in a well-ventilated environment and, if used inside a case, the case should not be covered.
- Whilst in use, this product should be placed on a stable, flat, non-conductive surface and should not be contacted by conductive items.
- The connection of incompatible devices to the GPIO connection may affect compliance, result in damage to the unit, and invalidate the warranty.
- All peripherals used with this product should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met. These articles include but are not limited to keyboards, monitors, and mice when used in conjunction with the Raspberry Pi.
- The cables and connectors of all peripherals used with this product must have adequate insulation so that relevant safety requirements are met.

Safety Instructions

To avoid malfunction of or damage to this product, please observe the following:

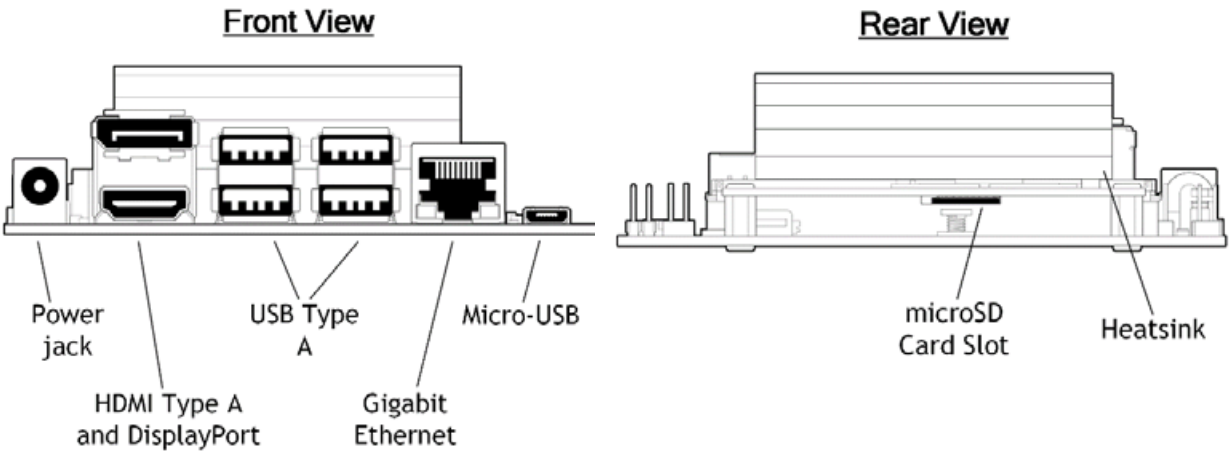
- Do not expose to water or moisture, or place on a conductive surface whilst in operation.
- Do not expose to heat from any source; the Raspberry Pi 3 Model B is designed for reliable operation at normal ambient temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Whilst it is powered, avoid handling the printed circuit board, or only handle it by the edges to minimize the risk of electrostatic discharge damage.

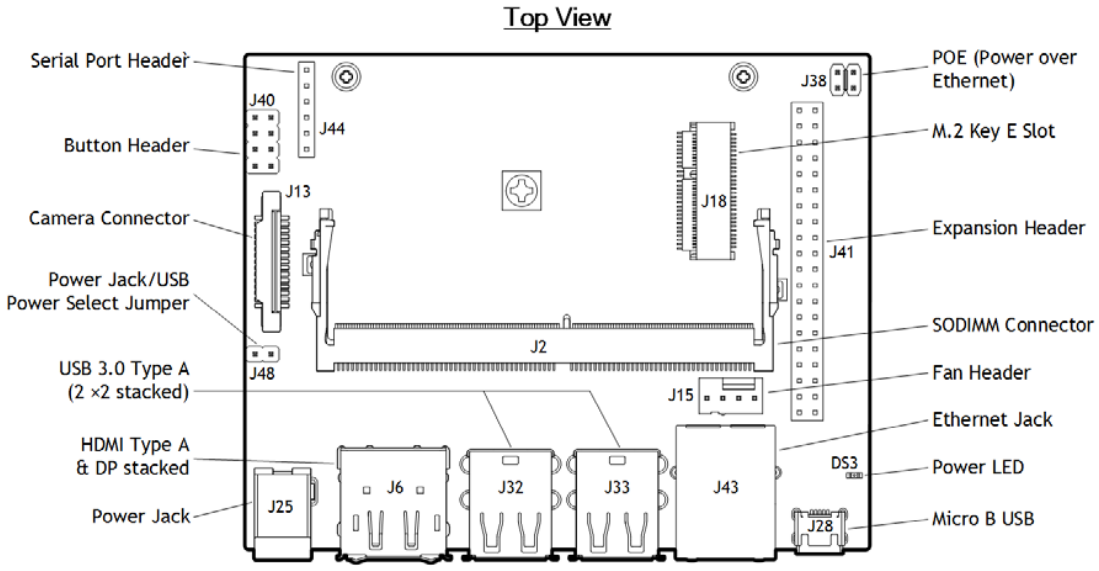
Appendix-E

NVIDIA Jetson Nano Datasheet and Jetpack

Developer Kit Interfaces

Developer kit module and carrier board





Jetpack

NVIDIA Jetpack SDK is the most comprehensive solution for building AI applications. It includes the latest OS images for Jetson products, along with libraries and APIs, samples, developer tools, and documentation. This section briefly describes each component of Jetpack

OS Image

Jetpack includes a reference file system derived from Ubuntu.

Libraries and APIs

Jetpack libraries and APIs include:

• TensorRT and cuDNN for high-performance deep learning applications

• CUDA for GPU accelerated applications across multiple domains

• NVIDIA Container Runtime for containerized GPU accelerated applications

• Multimedia API package for camera applications and sensor driver development

• VisionWorks, OpenCV, and VPI (Developer Preview) for visual computing applications

Published Papers

Paper 1 :

Title: Currency Detection for Visually Impaired Iraqi Banknote as a Study Case

Publisher : Turkish Journal of Computer and Mathematics Education

Index by Scopus Q4

Vol.12 No.6 (2021), 2940-2948

Article History: Received: 10 November 2020; Revised 12 January 2021
Accepted: 27 January 2021; Published online: 5 April 2021

Paper 2 :

Title: Real Time Object Detection for Visually Impaired Person

Publisher : Annals of the Romanian Society for Cell Biology

Index by Scopus Q4

ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 14725 – 14732
Article History: Received 05 March 2021; Accepted 01 April 2021

Paper 3 :

Title: Visually impaired assistance system

Publisher: المؤتمر الدولي العلمي الرابع المشترك بين الجامعة العراقية وجامعة دهوك

قبول نشر

العدد :149

التاريخ: 2021/1/17

TO /Raghad R Mahmood

السلام عليكم ورحمة الله وبركاته :

تدارست اللجنة العلمية للمؤتمر العلمي الدولي الرابع المشترك بين الجامعة العراقية وجامعة دهوك بالتعاون مع مركز نون للبحوث والدراسات المتخصصة والمنعقد تحت شعار

«أثر البحث العلمي في تحقيق اهداف التعليم»

للمدة من 16-17 كانون الاول 2020 البحث المقدم من قبلكم والموسوم:

(Visually impaired assistance system)

وبعد الاطلاع على آراء المقيمين، فقد قررت اللجنة قبول البحث،

وسينشر لاحقاً في مجلة الجامعة العراقية/عدد خاص

مع التقدير.....

أ.م.د نشوان محمود الصفار

رئيس المؤتمر

الخلاصة

وفقاً لسجلات منظمة الصحة العالمية ، هناك مايقارب 285 مليون شخص يعانون من إعاقة بصرية. من السهل نسبياً أن يتعامل الإنسان الطبيعي مع الأشياء الموجودة في البيئة المحيطة، ولكنها واحدة من المشكلات الرئيسية التي يواجهها الأشخاص المعاقين بصرياً، بالاضافة الى ذلك يواجه المعاقين بصرياً مشكلة كبيرة في التعرف على العملات النقدية التي تلعب دورا اساسيا في حياتنا اليومية.

تقترح الأطروحة الحالية نظام لمساعدة الأشخاص المعاقين بصرياً عن طريق تحويل العالم المرئي إلى أوامر صوتية. حيث تم تطوير النظام باستخدام التعلم العميق على أساس نماذج YOLOv3 ذات المرحلتين، دُرب كل نموذج على مجموعة بيانات محددة. تم بناء قاعدة البيانات المستخدمة في تدريب النماذج باستخدام نهج يركز على المريض تماماً، حيث تم اختيار أربعة وأربعين عنصرا للكشف عنهم. قُيم أداء النظام المقترح باستخدام مجموعة بيانات الاختبار وكذلك من خلال الكشف على الاشياء في فيديو مباشر من الكاميرا. أظهرت النتائج أن النظام المقترح يمكنه الكشف عن الأشياء والتعرف عليها بمتوسط دقة عالية تصل إلى ٪88.585 ودقة ٪97.647 للكشف عن الأوراق النقدية العراقية.

تم استخدام جهازين في تنفيذ نظام مساعدة ضعاف البصر. تم تنفيذ أول نظام مضمن على Raspberry Pi 3B والثاني على NVIDIA Jetson Nano.أظهرت نتائج تطبيق النظامين أن سرعة كشف NVIDIA Jetson Nano كانت أسرع بكثير من Raspberry Pi 3B.

اقرار لجنة المناقشة

نشهد بأننا أعضاء لجنة التقويم والمناقشة قد اطلعنا على هذه الرسالة الموسومة (**نظام مساعدة ضعاف البصر المبني بشبكة YOLOv3**) وناقشنا الطالبة (**رغد رائد محمود**) في محتوياتها وفيما له علاقة بها بتاريخ / / 2018 وقد وجدناها جدير بنيل شهادة الماجستير-علوم في اختصاص هندسة الحاسوب والمعلوماتية.

التوقيع:
رئيس اللجنة: أ.د.شفاء عبد الرحمن داؤد
التاريخ: / /2021

التوقيع:
عضو اللجنة: أ.م.د. محمد حازم يونس الجماس
التاريخ: / /2021

التوقيع:
عضو اللجنة: أ.م.د. محمد عبدالمطلب محمد عبدالله
التاريخ: / /2021

التوقيع:
عضو اللجنة (المشرف): د. ماجد ضرار يونس
التاريخ: / /2021

التوقيع:
عضو اللجنة (المشرف): د. عماد عطية خلف
التاريخ: / /2021

قرار مجلس الكلية

اجتمع مجلس كلية هندسة الالكترونيات بجلسته المنعقدة بتاريخ : / / 2021 وقرر المجلس منح الطالب شهادة الماجستير علوم في اختصاص هندسة الحاسوب والمعلوماتية.

مقرر المجلس: د.
التاريخ: / / 2021

رئيس مجلس الكلية:
التاريخ: / / 2021

إقرار المشرف

نشهد بان الرسالة الموسومة ب **" نظام مساعدة ضعاف البصر المبني بشبكة YOLOv3"** تمت تحت اشرافنا وهي جزء من متطلبات نيل شهادة الماجستير في هندسة الحاسوب والمعلوماتية

التوقيع: التوقيع:
المشرف : د. ماجد ضرار يونس المشرف : د. عماد عطية خلف
التاريخ: / / 2021

إقرار المقيم اللغوي

اشهد باني قمت بمراجعة الرسالة الموسومة ب **" نظام مساعدة ضعاف البصر المبني بشبكة YOLOv3"** من الناحية اللغوية وتصحيح ما ورد فيها من أخطاء لغوية وتعبيرية وبذلك أصبحت الرسالة مؤهلة للمناقشة بقدر تعلق الامر بسلامة الأسلوب وصحة التعبير.

التوقيع:
المقوم اللغوي: م.م.محمود أحمد محمود
التاريخ: / / 2021

إقرار رئيس لجنة الدراسات العليا

بناء على التوصيات المقدمة من قبل المشرف والمقوم اللغوي أرشح هذه الرسالة للمناقشة.
التوقيع:
الاسم: أ . م . معن أحمد شحاذة العدواني
التاريخ: / / 2021

إقرار رئيس القسم

بناء على التوصيات المقدمة من قبل المشرف والمقوم اللغوي ورئيس لجنة الدراسات العليا أرشح هذه الرسالة للمناقشة.
التوقيع:
الاسم: أ . م . معن أحمد شحاذة العدواني
التاريخ: / / 2021

نظام مساعدة ضعاف البصر
المبني بشبكة YOLOv3

رسالة تقدم بها

رغد رائد محمود

إلى
مجلس كلية هندسة الالكترونيات
جامعة نينوى
كجزء من متطلبات نيل شهادة الماجستير
في
هندسة الحاسوب والمعلوماتية

بإشراف

د. ماجد ضرار يونس

د. عماد عطية خلف

1443 هـ 2021م

جامعة نينوى

كلية هندسة الالكترونيات

قسم هندسة الحاسوب والمعلوماتية

نظام مساعدة ضعاف البصر المبني بشبكة YOLOv3

رغد رائد محمود

رسالة ماجستير علوم في هندسة الحاسوب والمعلوماتية

بإشراف

د. ماجد ضرار يونس
د. عماد عطية خلف

1443 هـ

2021 م